Should Ethereum be okay with enshrining more things in the protocol?

2023 Sep 30

Special thanks to Justin Drake, Tina Zhen and Yoav Weiss for feedback and review.

From the start of the Ethereum project, there was a strong philosophy of trying to make the core Ethereum as simple as possible, and do as much as possible by building protocols on top. In the blockchain space, the "do it on L1" vs "focus on L2s" debate is typically thought of [as being primarily about scaling](#), but in reality, similar issues exist for serving many kinds of Ethereum users' needs: digital asset exchange, privacy, usernames, advanced cryptography, account safety, censorship resistance, frontrunning protection, and the list goes on. More recently, however, there has been some cautious interest in being willing to enshrine more of these features into the core Ethereum protocol.

This post will go into some of the philosophical reasoning behind the original minimal-enshrinement philosophy, as well as some more recent ways of thinking about some of these ideas. The goal will be to start to build toward a framework for better identifying possible targets where enshrining certain features in the protocol might be worth considering.

Early philosophy on protocol minimalism

Early on in the history of what was then called "Ethereum 2.0", there was a strong desire to create a clean, simple and beautiful protocol that tried to do as little as possible itself, and left almost everything up to users to build on top. Ideally, the protocol would *just* be a virtual machine, and verifying a block would *just* be a single virtual machine call.
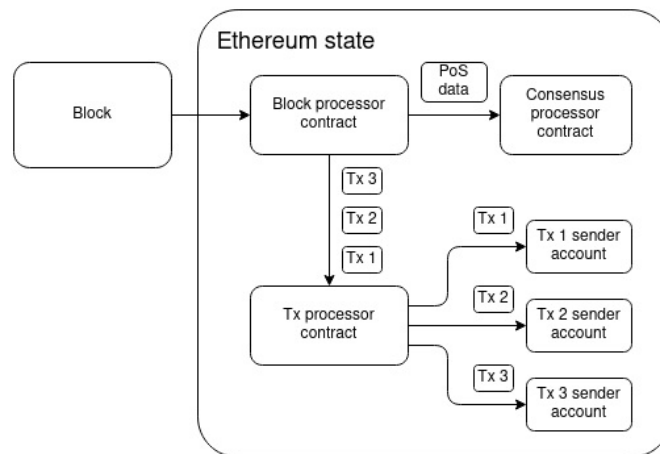
A very approximate reconstruction-from-memory of a whiteboard drawing Gavin Wood and I made back in early 2015, talking about what Ethereum 2.0 would look like.

The "state transition function" (the function that processes a block) would just be a single VM call, and all other logic would happen through contracts: a few system-level contracts, but mostly contracts provided by users. One really nice feature of this model is that even an entire hard fork could be described as a single transaction to the block processor contract, which would be approved through either offchain or onchain governance and then run with escalated permissions.

These discussions back in 2015 particularly applied to two areas that were on our minds: **account abstraction** and **scaling**. In the case of scaling, the idea was to try to create a maximally abstracted form of scaling that would feel like a natural extension of the diagram above. A contract could make a call to a piece of data that was not stored by most Ethereum nodes, and the protocol would detect that, and resolve the call through some kind of very generic scaled-computation functionality. From the virtual machine's point of view, the call would go off into some separate sub-system, and then some time later magically come back with the correct answer.

This line of thinking was explored briefly, but soon abandoned, because we were too preoccupied with verifying that any kind of blockchain scaling [was possible at all](#). Though as we will see later, the combination of [data availability sampling](#) and [ZK-EVMs](#) means that one possible future for Ethereum scaling might actually look surprisingly close to that vision! For account abstraction, on the other hand, we knew from the start that *some* kind of implementation was possible, and so research immediately began to try to make something as close as possible to the purist starting point of "a transaction is just a call" into reality.

```python
def apply_transaction(state, tx):
    state.logs = []
    state.suicides = []
    state.refunds = 0
    validate_transaction(state, tx)

    intrinsic_gas = tx.intrinsic_gas_used
    if state.is_HOMESTEAD():
        assert tx.s * 2 < transactions.secpk1n
        if not tx.to or tx.to == CREATE_CONTRACT_ADDRESS:
            intrinsic_gas += opcodes.CREATE[3]
            if tx.startgas < intrinsic_gas:
                raise InsufficientStartGas(rp(tx, 'startgas', tx.startgas, intrinsic_gas))

    log_tx.debug('TX NEW', txdict=tx.to_dict())

    # start transacting #################
    if tx.sender != null_address:
        state.increment_nonce(tx.sender)

    # buy startgas
    assert state.get_balance(tx.sender) >= tx.startgas * tx.gasprice
    state.delta_balance(tx.sender, -tx.startgas * tx.gasprice)

    message_data = vm.CallData([safe_ord(x) for x in tx.data], 0, len(tx.data))
    message = vm.Message(tx.sender, tx.to, tx.value, tx.startgas - intrinsic_gas, message_data, code_address=tx.to)

    # MESSAGE
    ext = VMExt(state, tx)
```

There is a lot of boilerplate code that occurs in between processing a transaction and making the actual underlying EVM call out of the sender address, and a lot more boilerplate that comes after. How do we reduce this code to as close to nothing as possible?

One of the major pieces of code in here is validate_transaction(state, tx), which does things like checking that the nonce and signature of the transaction are correct. The *practical* goal of account abstraction was, from the start, to allow the user to replace basic nonce-incrementing and ECDSA validation with their own validation logic, so that users could more easily use things like [social recovery and multisig wallets](). Hence, finding a way to rearchitect apply_transaction into just being a simple EVM call was not simply a "make the code clean for the sake of making the code clean" task; rather, it was about moving the logic into the user's account code, to give users that needed flexibility.

However, the insistence on trying to make apply_transaction contain as little enshrined logic as possible ended up introducing a lot of challenges. To see why, let us zoom in on one of the earliest account abstraction proposals, [EIP 86]():

Specification

If block.number >= METROPOLIS_FORK_BLKNUM, then: 1. If the signature of a transaction is (0, 0, 0) (ie. v = r = s = 0), then treat it as valid and set the sender address to 2**160 - 1 2. Set the address of any contract created through a creation transaction to equal sha3(0 + init code) % 2**160, where + represents concatenation, replacing the earlier address formula of sha3(rlp.encode([sender, nonce])) 3. Create a new opcode at 0xfb, CREATE_P2SH, which sets the creation address to sha3(sender + init code) % 2**160. If a contract at that address already exists, fails and returns 0 as if the init code had run out of gas.

Basically, if the signature is set to (0, 0, 0), then a transaction really does become "just a call". The account itself would be responsible for having code that parses the transaction, extracts and verifies the signature and nonce, and pays fees; see [here]() for an early example version of that code, and see [here]() for the very similar validate_transaction code that this account code would be replacing.

In exchange for this simplicity at protocol layer, miners (or, today, block proposers) gain the additional responsibility of running extra logic for only accepting and forwarding transactions that go to accounts whose code is set up to actually pay fees. What is that logic? Well, honestly EIP-86 did not think *too* hard about it:

Note that miners would need to have a strategy for accepting these transactions. This strategy would need to be very discriminating, because otherwise they run the risk of accepting transactions) for the validate_transaction code that this pre-account code would be replacingthat do not pay them any fees, and possibly even transactions that have no effect (eg. because the transaction was already included and so the nonce is no longer current). One simple approach is to have a whitelist for the codehash of accounts that they accept transactions being sent to; approved code would include logic that pays miners transaction fees.

However, this is arguably too restrictive; a looser but still effective strategy would be to accept any code that fits the same general format as the above, consuming only a limited amount of gas to perform nonce and signature checks and having a guarantee that transaction fees will be paid to the miner. Another strategy is to, alongside other approaches, try to process any transaction that asks for less than 250,000 gas, and include it only if the miner's balance is appropriately higher after executing the transaction than before it.

If EIP-86 had been included as-is, it would have reduced the complexity of the EVM, at the cost of massively increasing the complexity of other parts of the Ethereum stack, requiring essentially the exact same code to be written in other places, in addition to introducing entirely new classes of weirdness such as the possibility that the same transaction with the same hash might appear multiple times in the chain, not to mention the multi-invalidation problem.
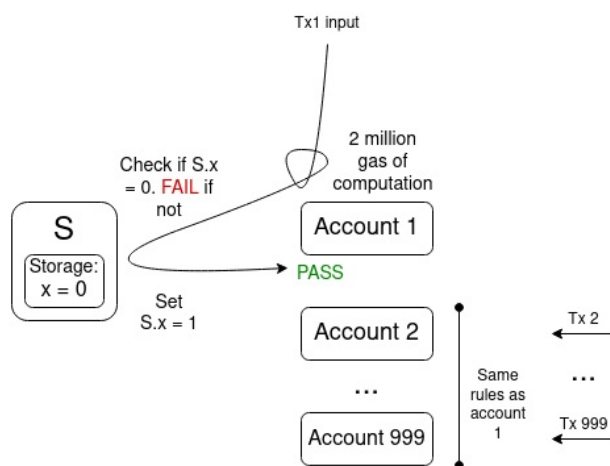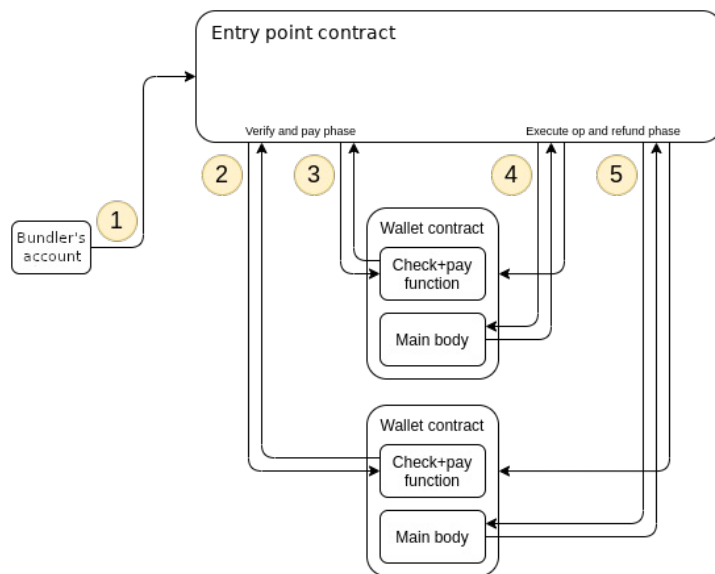

The multi-invalidation problem in account abstraction. One transaction getting included on chain could invalidate thousands of other transactions in the mempool, making the mempool easy to cheaply flood.

Acccount abstraction evolved in stages from there. EIP-86 became [EIP-208](#), which later became this [ethresear.ch post on "tradeoffs in account abstraction proposals"](#), which then became [this ethresear.ch post](#) half a year later. Eventually, out of all this, came the actually somewhat-workable [EIP-2938](#).

EIP-2938, however, *was not minimalistic at all*. The EIP includes:

- A new transaction type
- Three new transaction-wide global variables
- Two new opcodes, including the highly unwieldy PAYGAS opcode that handles gas price and gas limit checking, being an EVM execution breakpoint, and temporarily storing ETH for fee payments all at once.
- A set of complex mining and rebroadcasting strategies, including a list of banned opcodes for the validation phase of a transaction

In order to get account abstraction off the ground without involving Ethereum core developers who were busy on heroic efforts optimizing the Ethereum clients and implementing the merge, EIP-2938 eventually was rearchitected into the entirely [extra-protocol ERC-4337](#).

Entry point contract

Verify and pay phase   Execute op and refund phase

Bundler's account

Wallet contract
Check+pay function
Main body

Wallet contract
Check+pay function
Main body

1 2 3 4 5

ERC-4337. It really does rely entirely on EVM calls for everything!

Because it's an ERC, it does not require a hard fork, and technically lives "outside of the Ethereum protocol". So.... problem solved? Well, as it turns out, not quite. The *current* medium-term roadmap for ERC-4337 actually does involve eventually turning large parts of ERC-4337 into a series of protocol features, and it's a useful instructive example to see the reasons why this path is being considered.

Enshrining ERC-4337

There have been a few key reasons discussed for eventually bringing ERC-4337 back into the protocol:

- **Gas efficiency**: Anything done inside the EVM incurs some level of virtual machine overhead, including inefficiency in how it uses gas-expensive features like storage slots. Currently, these extra inefficiencies add up to at least ~20,000 gas, and often more. Pushing these components into the protocol is the easiest way to remove these issues.
- **Code bug risk**: if the ERC-4337 "entry point contract" has a sufficiently terrible bug, *all* ERC-4337-compatible wallets could see all of their funds drained. Replacing the contract with an in-protocol functionality creates an implied responsibility to fix code bugs with a hard fork, which removes funds-draining risk for users.
- **Support for EVM opcodes like `tx.origin`**. ERC-4337, by itself, makes `tx.origin` return the address of the "bundler" that packaged up a set of user operations into a transaction. Native account abstraction could fix this, by making `tx.origin` point to the actual account sending the transaction, making it work the same way as for EOAs.
- **Censorship resistance**: one of the challenges with proposer/builder separation is that it becomes easier to censor individual transactions. In a world where individual *transactions* are legible to the Ethereum protocol, this problem can be greatly mitigated with inclusion lists, which allow proposers to specify a list of transactions that *must* be included within the next two slots in almost all cases. But the extra-protocol ERC-4337 wraps "user operations" inside a single transaction, making user operations opaque to the Ethereum protocol; hence, Ethereum-protocol-provided inclusion lists would not be able to provide censorship resistance to ERC-4337 user operations. Enshrining ERC-4337, and making user operations a "proper" transaction type, would solve this problem.

It's worth zooming into the gas efficiency issue further. In its current form, ERC-4337 is significantly more expensive than a "basic" Ethereum transaction: the transaction costs 21,000 gas, whereas ERC-4337 costs ~42,000 gas. This doc lists some of the reasons why:

- Need to pay lots of individual storage read/write costs, which in the case of EOAs get bundled into a single 21000 gas payment:
  - Editing the storage slot that contains pubkey+nonce (~5000)
  - UserOperation calldata costs (~4500, reducible to ~2500 with compression)
  - ECRECOVER (~3000)
  - Warming the wallet itself (~2600)
  - Warming the recipient account (~2600)
  - Transferring ETH to the recipient account (~9000)
  - Editing storage to pay fees (~5000)
  - Access the storage slot containing the proxy (~2100) and then the proxy itself (~2600)
- On top of the above storage read/write costs, the contract needs to do "business logic" (unpacking the UserOperation, hashing it, shuffling variables, etc) that EOA transactions have handled "for free" by the Ethereum protocol
- Need to expend gas to pay for logs (EOAs don't issue logs)
- One-time contract creation costs (~32000 base, plus 200 gas per code byte in the proxy, plus 20000 to set the proxy address)

Theoretically, it should be possible to massage the EVM gas cost system until the in-protocol costs and the extra-protocol costs for accessing storage match; there is no reason why transferring ETH needs to cost 9000 gas when other kinds of storage-editing operations are much cheaper. And indeed, two EIPs ([1] [2]) related to the upcoming Verkle tree transition actually try to do that. But even if we do that, there is one huge reason why enshrined protocol features are going to inevitably be significantly cheaper than EVM code, no matter how efficient the EVM becomes: **enshrined code does not need to pay gas for being pre-loaded**.

Fully functional ERC-4337 wallets are *big*. This implementation, compiled and put on chain, takes up ~12,800 bytes. Of course, you can deploy that big piece of code once, and use DELEGATECALL to allow each individual wallet to call into it, but that code still needs to be accessed in each block that uses it. Under the Verkle tree gas costs EIP, 12,800 bytes would make up 413 chunks, and accessing those chunks would require paying 2x WITNESS_BRANCH_COST (3,800 gas total) and 413x WITNESS_CHUNK_COST (82,600 gas total). And this does not even begin to mention the ERC-4337 entry-point itself, with 23,689 bytes onchain in version 0.6.0 (under the Verkle tree EIP rules, ~158,700 gas to load).

This leads to a problem: the gas costs of actually accessing this code would have to be split among transactions somehow. The current approach that ERC-4337 uses is not great: the first transaction in a bundle eats up one-time storage/code reading costs, making it much more expensive than the rest of the transactins. Enshrinement in-protocol would allow these commonly-shared libraries to simply be part of the protocol, accessible to all with no fees.

What can we learn from this example about when to enshrine things more generally?

In this example, we saw a few different rationales for enshrining aspects of account abstraction in the protocol.

- **"Move complexity to the edges" market-based approaches break down the most when there are high fixed costs**. And indeed, the long term account abstraction roadmap looks like it's going to have *lots* of fixed costs per block. 244,100 gas for loading standardized wallet code is one thing; but aggregation (see my presentation from this summer for more details) potentially adds hundreds of thousands more gas for ZK-SNARK validation plus onchain costs for proof verification. There isn't a way to charge users for these costs without introducing lots of market inefficiencies, whereas making some of these functionalities into protocol features accessible to all with no fees cleanly solves that problem.
- **Community-wide response to code bugs**. If some set of pieces of code are used by all users, or a very wide class of users, then it often makes more sense for the blockchain community to take on itself the responsibility to hard-fork to fix any bugs that arise. ERC-4337 introduced a large amount of globally shared code, and in the long term it makes more sense for bugs in that code to be fixed by hard forks than to lead to users losing a large amount of ETH.
- **Sometimes, a stronger form of a feature can be implemented by directly taking advantage of the powers of the protocol**. The key example here is in-protocol censorship resistance features like inclusion lists: in-protocol inclusion lists can do a better job of guaranteeing censorship resistance than extra-protocol approaches, in order for user-level operations to actually benefit from in-protocol inclusion lists, individual user-level operations need to be "legible" to the protocol. Another lesser-known example is that 2017-era Ethereum proof of stake designs had account abstraction for staking keys, and this was abandoned in favor of enshrining BLS because BLS supported an "aggregation" mechanism, which would have to be implemented at protocol and network level, that could make handling a very large number of signatures much more efficient.

But it is important to remember that **even enshrined in-protocol account abstraction is still a massive "de-enshrinement" compared to the status quo.** Today, top-level Ethereum transactions can only be initiated from externally owned accounts (EOAs) which use a single secp256k1 elliptic curve signature for verification. Account abstraction de-enshrines this, and leaves verification conditions open for users to define. And so, in this story about account abstraction, we also saw the biggest argument *against* enshrinement: **being flexible to diverse users' needs**.
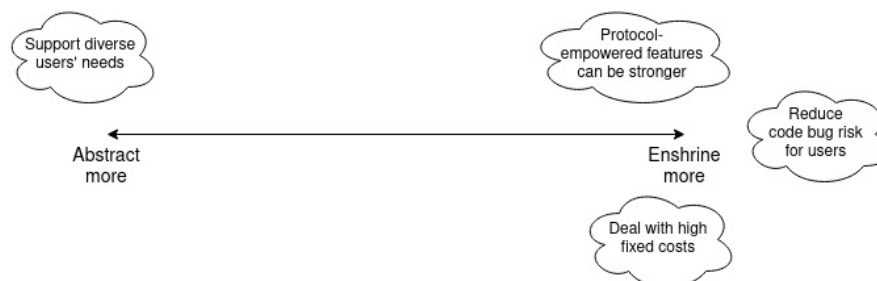

Let us try to fill in the story further, by looking at a few other examples of features that have recently been considered for enshrinement. We'll particularly focus on: **ZK-EVMs**, **proposer-builder separation**, **private mempools**, **liquid staking** and **new precompiles**.

Enshrining ZK-EVMs

Let us switch focus to another potential target for enshrining into the Ethereum protocol: ZK-EVMs. Currently, we

have a large number of [ZK-rollups](#) that all have to write fairly similar code to [verify execution of Ethereum-like blocks inside a ZK-SNARK](#). There is a pretty diverse ecosystem of independent implementations: [the PSE ZK-EVM](#), [Kakarot](#), [the Polygon ZK-EVM](#), [Linea](#), [Zeth](#), and the list goes on.

One of the recent controversies in the EVM ZK-rollup space has to do with how to deal with the possibility of bugs in the ZK-code. Currently, all of these systems that are live have some form of "security council" mechanism that can override the proving system in case of a bug. In [this post last year](#), I tried to create a standardized framework to encourage projects to be clear about what level of trust they put in the proving system and what level in the security council, and move toward giving less and less powers to the security council over time.
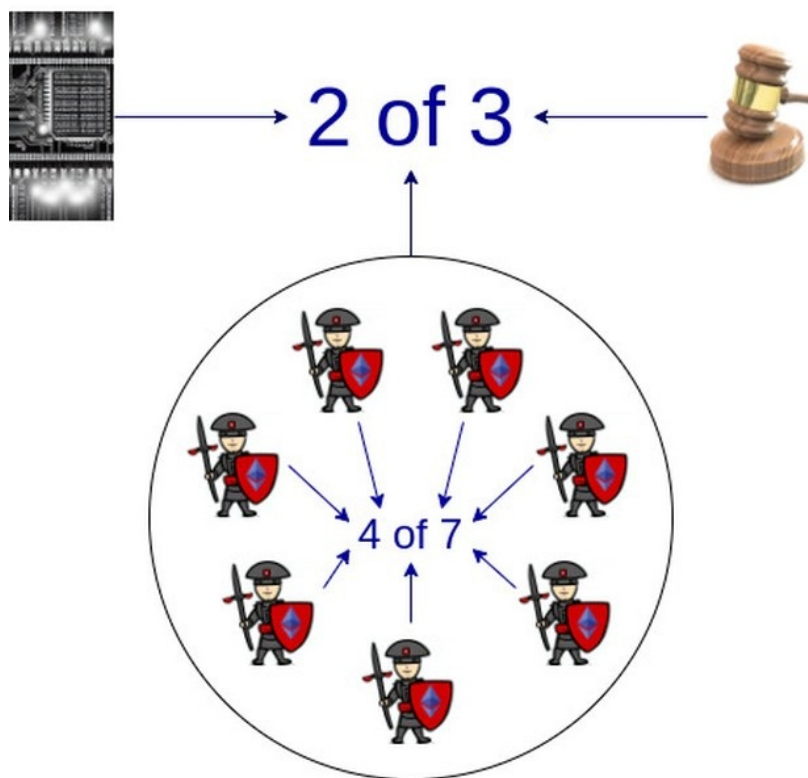
In the medium term, rollups could rely on [multiple proving systems](#), and the security council would only have any power at all in the extreme case where two different proving systems disagree with each other.

However, there is a sense in which some of this work *feels superfluous*. We already have the Ethereum base layer, which has an EVM, and we already have a working mechanism for dealing with bugs in implementations: if there's a bug, the clients that have the bug update to fix the bug, and the chain goes on. Blocks that appeared finalized from the perspective of a buggy client would end up no-longer-finalized, but at least we would not see users losing funds. Similarly, if a rollup just wants to be and remain EVM-equivalent, it feels wrong that they need to implement their own governance to keep changing their internal ZK-EVM rules to match upgrades to the Ethereum base layer, when ultimately they're building on top of the Ethereum base layer itself, which knows when it's being upgraded and to what new rules.

Since these L2 ZK-EVMs are basically using the exact same EVM as Ethereum, can't we somehow make "verify EVM execution in ZK" into a protocol feature, and deal with exceptional situations like bugs and upgrades by just applying Ethereum's social consensus, the same way we already do for base-layer EVM execution itself?

This is an important and challenging topic. There are a few nuances:

1. **We want to be [compatible with Ethereum's multi-client philosophy](#)**. This means that we want to allow different clients to use different proving systems. This in turn implies that for any EVM execution that gets proven with one ZK-SNARK system, we want a guarantee that the underlying data is *[available](#)*, so that proofs can be generated for other ZK-SNARK systems.
2. **While the tech is immature, we probably want auditability.** In practice, this means the exact same thing: if any execution gets proven, we want the underlying data to be available, so that if anything goes wrong, users and developers can inspect it.
3. **We need much faster proving times**, so that if one type of proof is made, other types of proof can be generated quickly enough that other clients can validate them. One *could* get around this by making a precompile that has an asynchronous response after some time window longer than a slot (eg. 3 hours), but this adds complexity.
4. **We want to support not just copies of the EVM, but also "almost-EVMs".** Part of the attraction of L2s is the ability to innovate on the execution layer, and make extensions to the EVM. If a given L2's VM differs from the EVM only a little bit, it would be nice if the L2 could still use a native in-protocol ZK-EVM for the parts that are identical to the EVM, and only rely on their own code for the parts that are different. This could be done by

designing the ZK-EVM precompile in such a way that it allows the caller to specify a bitfield or list of opcodes or addresses that get handled by an externally supplied table instead of the EVM itself. We could also make gas costs open to customization to a limited extent.

One likely topic of contention with data availability in a native ZK-EVM is **statefulness**. ZK-EVMs are much more data-efficient if they do not have to carry "witness" data. That is, if a particular piece of data was already read or written in some previous block, we can simply assume that provers have access to it, and we don't have to make it available again. This goes beyond not re-loading storage and code; it turns out that if a rollup properly *compresses* data, the compression being *stateful* allows for up to 3x data savings compared to the compression being stateless.
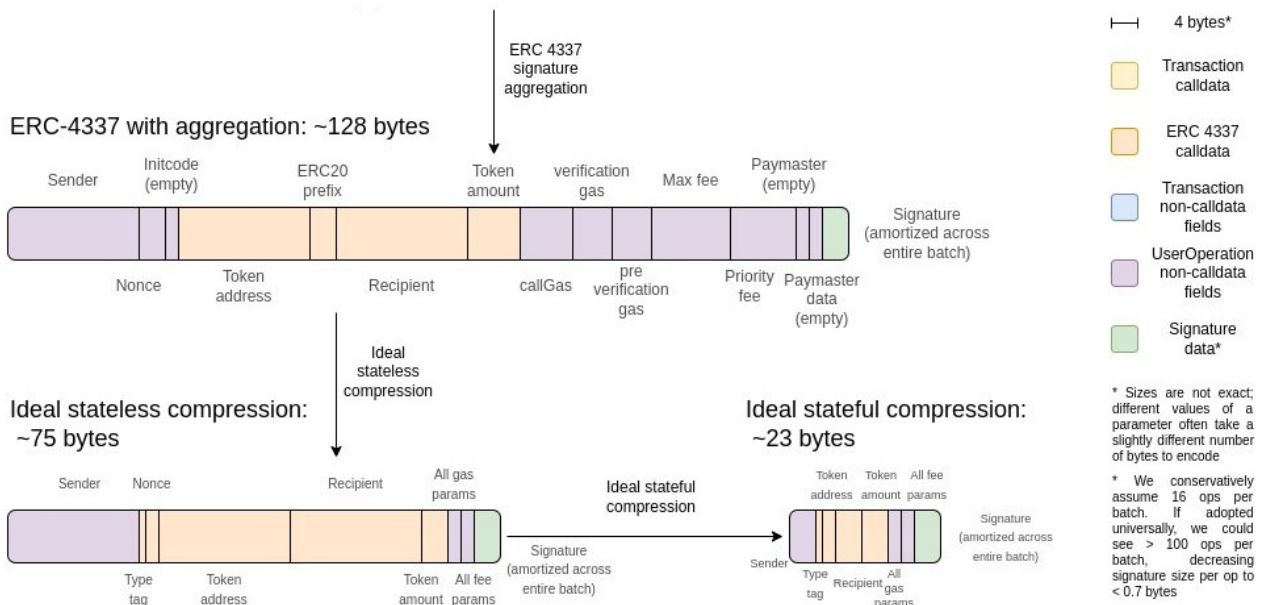
This means that for a ZK-EVM precompile, we have two options:

1. **The precompile requires all data to be available in the same block**. This means that provers can be stateless, but it also means that ZK-rollups using such a precompile become much more expensive than rollups using custom code.
2. **The precompile allows pointers to data used or generated by *previous* executions.** This allows ZK-rollups to be near-optimal, but it's more complicated and introduces a new kind of state that has to be stored by provers.

What lessons can we take away from this? There is a pretty good argument to enshrine ZK-EVM validation somehow: rollups are already building their own custom versions of it, and it *feels wrong* that Ethereum is willing to put the weight of its multiple implementations and off-chain social consensus behind EVM execution on L1, but L2s doing the exact same work have to instead implement complicated gadgets involving security councils. But on the other hand, there is a big devil in the details: there are different versions of an enshrined ZK-EVM that have different costs and benefits. The stateful vs stateless divide only scratches the surface; attempting to support "almost-EVMs" that have custom code proven by other systems will likely reveal an even larger design space. Hence, **enshrining ZK-EVMs presents both promise and challenges**.

Enshrining proposer-builder separation (ePBS)

The rise of [MEV](#) has made block production into an economies-of-scale-heavy activity, with sophisticated actors being able to produce blocks that generate much more revenue than default algorithms that simply watch the mempool for transactions and include them. The Ethereum community has so far attempted to deal with this by using *extra-protocol proposer-builder separation* schemes like [MEV-Boost](#), which allow regular validators ("proposers") to outsource block building to specialized actors ("builders").

However, MEV-Boost carries a trust assumption in a new category of actor, called a *relay*. For the past two years, there have been [many proposals to create "enshrined PBS"](#). What is the benefit of this? In this case, the answer is pretty simple: the PBS that can be built by directly using the powers of the protocol is simply stronger (in the sense of having weaker trust assumptions) than the PBS that can be built without them. It's a similar case to the case for [enshrining in-protocol price oracles](#) - though, in *that* situation, there is also a [strong counterargument](#).

Enshrining private mempools

When a user sends a transaction, that transaction becomes immediately public and visible to all, even before it gets included on chain. This makes users of many applications vulnerable to economic attacks such as frontrunning: if a user makes a large trade on eg. Uniswap, an attacker could put in a transaction right before them, increasing the price at which they buy, and collecting an arbitrage profit.

Recently, there has been a number of projects specializing in creating "private mempols" (or "encrypted mempools"), which keep users' transactions encrypted until the moment they get irreversibly accepted into a block.

The problem is, however, that schemes like this require a particular kind of encryption: to prevent users from flooding the system and frontrunning the decryption process *itself*, the encryption must auto-decrypt once the transaction actually does get irreversibly accepted.

To implement such a form of encryption, there are various different technologies with different tradeoffs, described well in [this post by Jon Charbonneau](#) (and [this video](#) and [slides](#)):

- **Encryption to a centralized operator**, eg. [Flashbots Protect](#).
- **Time-lock encryption**, a form of encryption which can be decrypted by anyone after a certain number of *sequential* computational steps, which cannot be parallelized.
- **Threshold encryption**, trusting an honest majority committee to decrypt the data. See the [shutterized beacon chain](#) concept for a concrete proposal.
- **Trusted hardware** such as [SGX](#).

Unfortunately, each of these have varying weaknesses. A centralized operator is not acceptable for inclusion in-protocol for obvious reasons. Traditional time lock encryption is too expensive to run across thousands of transactions in a public mempool. A more powerful primitive called **delay encryption** allows efficient decryption of an unlimited number of messages, but it's hard to construct in practice, and [attacks on existing constructions](#) still sometimes get discovered. Much like [with hash functions](#), we'll likely need a period of more years of research and analysis before delay encryption becomes sufficiently mature. Threshold encryption requires trusting a majority to not collude, in a setting where they can collude *undetectably* (unlike 51% attacks, where it's immediately obvious who participated). SGX creates a dependency on a single trusted manufacturer.

While for each solution, there is some subset of users that is comfortable trusting it, there is no single solution that is trusted enough that it can practically be accepted into layer 1. Hence, enshrining anti-frontrunning at layer 1 seems like a difficult proposition at least until delay encrypted is perfected or there is some other technological breakthrough, even while it's a valuable enough functionality that lots of application solutions will already emerge.

Enshrining liquid staking

A common demand among Ethereum defi users is the ability to use their ETH for staking and as collateral in other applications at the same time. Another common demand is simply for convenience: users want to be able to stake without the complexity of running a node and keeping it online all the time (and protecting their now-online staking keys).

By far the simplest possible "interface" for staking, which satisfies both of these needs, is just an ERC20 token: convert your ETH into "staked ETH", hold it, and then later convert back. And indeed, liquid staking providers such as [Lido](#) and [Rocketpool](#) have emerged to do just that. However, liquid staking has some natural centralizing mechanics at play: people naturally go into the biggest version of staked ETH because it's most familiar and most liquid (and most well-supported by applications, who in turn support it because it's more familiar and because it's the one the most users will have heard of).

Each version of staked ETH needs to have some mechanism determining who can be the underlying node operators. It can't be unrestricted, because then attackers would join and amplify their attacks with users' funds. Currently, the top two are Lido, which has a DAO whitelisting node operators, and Rocket Pool, which allows anyone to run a node if they [put down 8 ETH](#) (ie. 1/4 of the capital) as a deposit. These two approaches have different risks: the Rocket Pool approach allows attackers to 51% attack the network, and force users to pay most of the costs. With the DAO approach, if a single such staking token dominates, that leads to a single, [potentially](#) [attackable](#) governance gadget controlling a very large portion of all Ethereum validators. To the credit of protocols like Lido, they have [implemented safeguards against this](#), but one layer of defense may not be enough.

In the short term, one option is to socially encourage ecosystem participants to use a diversity of liquid staking providers, to reduce the chance that any single one becomes too large to be a systemic risk. In the longer term, however, this is an unstable equilibrium, and there is peril in relying too much on moralistic pressure to solve problems. One natural question arises: **might it make sense to enshrine some kind of in-protocol functionality to make liquid staking less centralizing**?

Here, the key question is: *what kind of in-protocol functionality*? Simply creating an in-protocol fungible "staked ETH" token has the problem that it would have to either have an enshrined Ethereum-wide governance to choose who runs the nodes, or be open-entry, turning it into a vehicle for attackers.

One interesting idea is [Dankrad Feist's writings on liquid staking maximalism](). First, we bite the bullet that if Ethereum gets 51% attacked, only perhaps 5% of the attacking ETH gets slashed. This is a reasonable tradeoff; right now there is [over 26 million ETH being staked](), and a cost of attack of 1/3 of that (~8 million ETH) is way overkill, especially considering how many kinds of "outside-the-model" attacks can be pulled off for much less. Indeed, a similar tradeoff has already been explored in the ["super-committee" proposal]() for implementing single-slot finality.
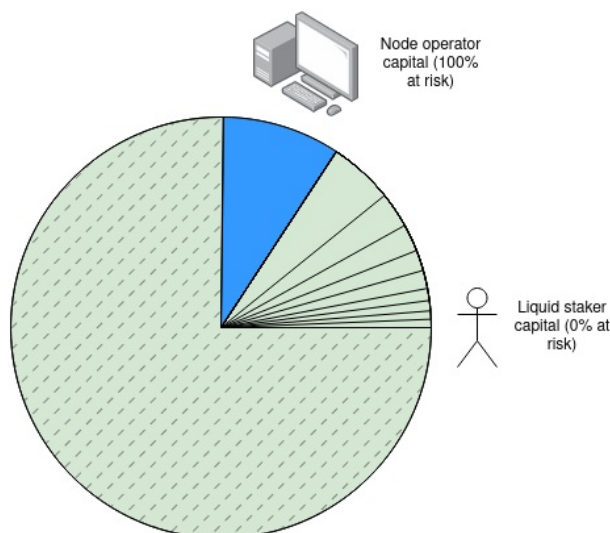
If we accept that only 5% of attacking ETH gets slashed, then over 90% of staked ETH would be invulnerable to slashing, and so 90% of staked ETH could be put into an in-protocol fungible liquid staking token that can then be used by other applications.

This path is interesting. But it still leaves open the question: **what is the specific thing that would get enshrined**? [RocketPool]() already works in a way very similar to this: each node operator puts up some capital, and liquid stakers put up the rest. We could simply tweak a few constants, bounding the maximum slashing penalty to eg. 2 ETH, and Rocket Pool's *existing* rETH would become risk-free.

There are other clever things that we can do with simple protocol tweaks. For example, imagine that we want a system where there are [two "tiers" of staking](): node operators (high collateral requirement) and depositors (no minimum, can join and leave any time), but we still want to guard against node operator centralization by giving a randomly-sampled committee of depositors powers like suggesting lists of transactions that have to be included (for anti-censorship reasons), controlling the fork choice during an inactivity leak, or needing to sign off on blocks. This could be done in a mostly-out-of-protocol way, by tweaking the protocol to require each *validator* to provide (i) a regular staking key, and (ii) an ETH address that can be called to output a secondary staking key during each slot. The protocol would give powers to these two keys, but the mechanism for choosing the second key in each slot could be left to staking pool protocols. It may still be better to enshrine some things outright, but it's valuable to note that this "enshrine some things, leave other things to users" design space exists.

Enshrining more precompiles

[Precompiles]() (or "precompiled contracts") are Ethereum contracts that implement complex cryptographic operations, whose logic is natively implemented in client code, instead of EVM smart contract code. Precompiles were a compromise adopted at the beginning of Ethereum's development: because the overhead of a VM is too much for certain kinds of very complex and highly specialized code, we can implement a few key operations valuable to important kinds of applications in native code to make them faster. Today, this basically includes a few specific [hash functions and elliptic curve operations]().

There is currently a push to add [a precompile for secp256r1](), an elliptic curve slightly different from the secp256k1 used for basic Ethereum accounts, because it is well-supported by trusted hardware modules and thus widespread use of it could improve wallet security. In recent years, there have also been pushes to add precompiles for [BLS-12-377](), [BW6-761](), [generalized pairings]() and other features.

The counterargument to these requests for more precompiles is that many of the precompiles that have been added before (eg. [RIPEMD]() and [BLAKE]()) have ended up gotten [used much less than anticipated](), and we should learn from

that. Instead of adding more precompiles for specific operations, we should perhaps focus on a more moderate approach based on ideas like EVM-MAX and the dormant-but-always-revivable SIMD proposal, which would allow EVM implementations to execute wide classes of code less expensively. Perhaps even *existing* little-used precompiles could be removed and replaced with (unavoidably less efficient) EVM code implementations of the same function. That said, it is still possible that there are specific cryptographic operations that are valuable enough to accelerate that it makes sense to add them as precompiles.

What do we learn from all this?

The desire to enshrine as little as possible is understandable and good; it hails from the Unix philosophy tradition of creating software that is minimalist and can be easily adapted to different needs by its users, avoiding the curses of software bloat. However, **blockchains are not personal-computing operating systems; they are social systems**. This means that there are rationales for enshrining certain features in the protocol that go beyond the rationales that exist in a purely personal-computing context.

In many cases, these other examples re-capped similar lessons to what we saw in account abstraction. But there are also a few new lessons that have been learned as well:

- **Enshrining features can help avoid centralization risks in other areas of the stack**. Often, keeping the base protocol minimal and simple pushes the complexity to some outside-the-protocol ecosystem. From a Unix philosophy perspective, this is good. Sometimes, however, there are risks that that outside-the-protocol ecosystem will centralize, often (but not just) because of high fixed costs. Enshrining can sometimes decrease de-facto centralization.
- **Enshrining too much can over-extend the trust and governance load of the protocol**. This is the topic of this earlier post about not overloading Ethereum's consensus: if enshrining a particular feature weakens the trust model, and makes Ethereum as a whole much more "subjective", that weakens Ethereum's credible neutrality. In those cases, it's better to leave that particular feature as a mechanism on top of Ethereum, and not try to bring it inside Ethereum itself. Here, encrypted mempools are the best example of something that may be a bit too difficult to enshrine, at least until/unless delay encryption technology improves.
- **Enshrining too much can over-complicate the protocol**. Protocol complexity is a systemic risk, and adding too many features in-protocol increases that risk. Precompiles are the best example of this.
- **Enshrining can backfire in the long term, as users' needs are unpredictable**. A feature that many people *think* is important and will be used by many users may well turn out not to be used much in practice.

Additionally, the liquid staking, ZK-EVM and precompile cases show the possibility of a middle road: **minimal viable enshrinement**. Rather than enshrining an entire functionality, the protocol could enshrine a specific piece that solves the key challenges with making that functionality easy to implement, without being *too* opinionated or narrowly focused. Examples of this include:

- Rather than enshrining a full liquid staking system, changing staking penalty rules to make trustless liquid staking more viable
- Rather than enshrining more precompiles, enshrine EVM-MAX and/or SIMD to make a wider class of operations simpler to implement efficiently
- Rather than enshrining the whole concept of *rollups*, we could simply enshrine EVM *verification*.

We can extend our diagram from earlier in the post as follows:
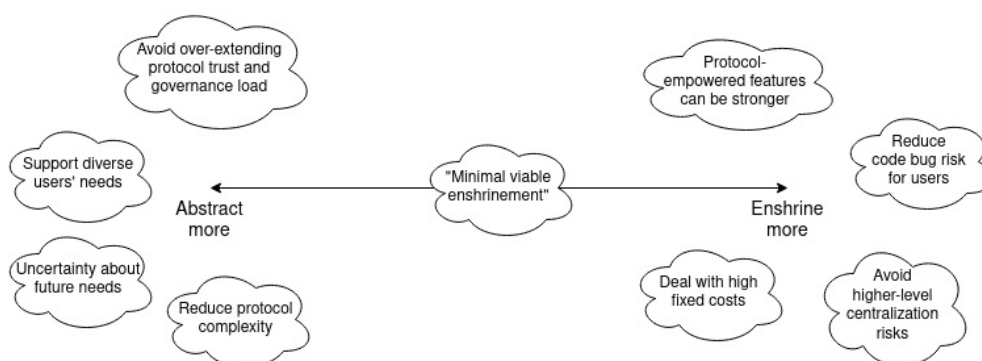
Sometimes, it may even make sense to *de-enshrine* a few things. De-enshrining little-used precompiles is one example. Account abstraction as a whole, as mentioned earlier, is also a significant form of de-enshrinement. If we want to support backwards-compatibility for existing users, then the mechanism may actually be surprisingly similar to that for de-enshrining precompiles: one of the proposals is EIP-5003, which would allow EOAs to convert their account in-plce into a contract that has the same (or better) functionality.

What features should be brought into the protocol and what features should be left to other layers of the ecosystem is a complicated tradeoff, and we should expect the tradeoff to continue to evolve over time as our understanding of users' needs and our suite of available ideas and technologies continues to improve.

What do I think about Community Notes?

Special thanks to Dennis Pourteaux and Jay Baxter for feedback and review.

The last two years of ~~Twitter~~ X have been tumultuous, to say the least. After the platform was ~~bought not bought~~ bought by Elon Musk for $44 billion last year, Elon enacted sweeping changes to the company's staffing, content moderation and business model, not to mention changes to the culture on the site that may well have been a result of Elon's soft power more than any specific policy decision. But in the middle of these highly contentious actions, one new feature on Twitter grew rapidly in importance, and seems to be beloved by people across the political spectrum: Community Notes.
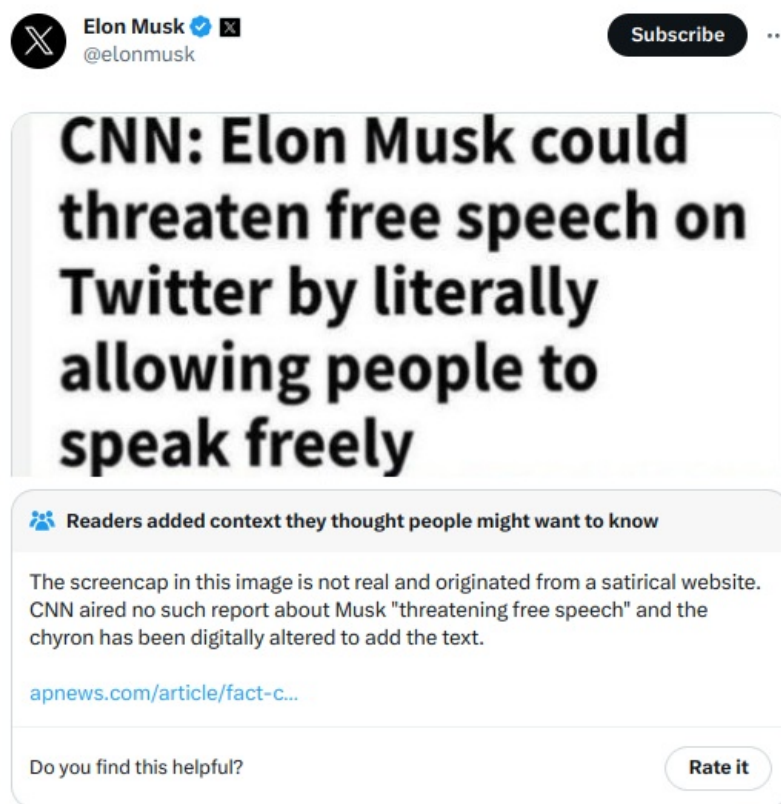
Community Notes is a fact-checking tool that sometimes attaches context notes, like the one on Elon's tweet above, to tweets as a fact-checking and anti-misinformation tool. It was originally called Birdwatch, and was first rolled out as a pilot project in January 2021. Since then, it has expanded in stages, with the most rapid phase of its expansion coinciding with Twitter's takeover by Elon last year. Today, Community Notes appear frequently on tweets that get a very large audience on Twitter, including those on contentious political topics. And both in my view, and in the view of many people across the political spectrum I talk to, the notes, when they appear, are informative and valuable.

But what interests me most about Community Notes is how, despite not being a "crypto project", it might be the closest thing to an instantiation of "crypto values" that we have seen in the mainstream world. Community Notes are not written or curated by some centrally selected set of experts; rather, they can be written and voted on by anyone, and which notes are shown or not shown is decided entirely by an open source algorithm. The Twitter site has a detailed and extensive guide describing how the algorithm works, and you can download the data containing which notes and votes have been published, run the algorithm locally, and verify that the output matches what is visible on the

Twitter site. It's not perfect, but it's surprisingly close to satisfying the ideal of [credible neutrality](#), all while being impressively useful, even under contentious conditions, at the same time.

How does the Community Notes algorithm work?

Anyone with a Twitter account [matching some criteria](#) (basically: active for 6+ months, no recent rule violations, verified phone number) can sign up to participate in Community Notes. Currently, participants are slowly and randomly being accepted, but eventually the plan is to let in anyone who fits the criteria. Once you are accepted, you can at first participate in rating existing notes, and once you've made enough good ratings ([measured by](#) seeing which ratings match with the final outcome for that note), you can also write notes of your own.

When you write a note, the note gets a score based on the reviews that it receives from other Community Notes members. These reviews can be thought of as being votes along a 3-point scale of HELPFUL, SOMEWHAT_HELPFUL and NOT_HELPFUL, but a review can also contain some other tags that have roles in the algorithm. Based on these reviews, a note gets a score. If the note's score is above 0.40, the note is shown; otherwise, the note is not shown.

The way that the score is calculated is what makes the algorithm unique. Unlike simpler algorithms, which aim to simply calculate some kind of sum or average over users' ratings and use that as the final result, **the Community Notes rating algorithm explicitly attempts to prioritize notes that receive positive ratings from people across a diverse range of perspectives**. That is, if people who usually disagree on how they rate notes end up agreeing on a particular note, that note is scored especially highly.

Let us get into the deep math of how this works. We have a set of users and a set of notes; we can create a matrix M, where the cell $M_{i,j}$ represents how the i'th user rated the j'th note.
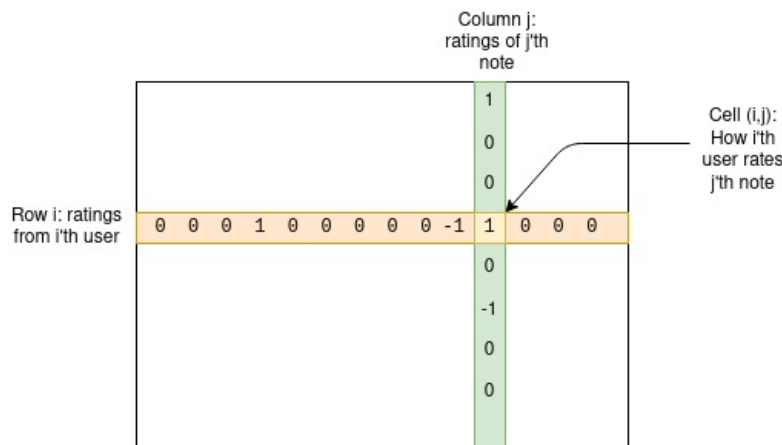
For any given note, most users have not rated that note, so most entries in the matrix will be zero, but that's fine. The goal of the algorithm is to create a four-column model of users and notes, assigning each user two stats that we can call "friendliness" and "polarity", and each note two stats that we can call "helpfulness" and "polarity". The model is trying to predict the matrix as a function of these values, using the following formula:

$$M_{un} \approx \mu + i_u + i_n + f_u * f_n$$

Prediction of the matrix entry at row (user) **u**, column (note) **n**	"General public mood"	User **u**'s friendliness ("intercept")	Note **n**'s helpfulness ("intercept")	User **u**'s polarity ("factor")	Note **n**'s polarity ("factor")

Note that here I am introducing both the terminology used in the [Birdwatch paper](#), and my own terms to provide a less mathematical intuition for what the variables mean:

- μ is a "**general public mood**" parameter that accounts for how high the ratings are that users give in general
- i_u is a user's "**friendliness**": how likely that particular user is to give high ratings
- i_n is a note's "**helpfulness**": how likely that particular note is to get rated highly. **Ultimately, this is the variable we care about**.
- f_u or f_n is user or note's "**polarity**": its position among the dominant axis of political polarization. In practice, negative polarity roughly means "left-leaning" and positive polarity means "right-leaning", but note that **the axis of polarization is discovered emergently from analyzing users and notes; the concepts of leftism and rightism are in no way hard-coded**.

The algorithm uses a pretty basic machine learning model (standard [gradient descent](#)) to find values for these variables that do the best possible job of predicting the matrix values. The helpfulness that a particular note is assigned is the note's final score. If a note's helpfulness is at least +0.4, the note gets shown.

The core clever idea here is that the "polarity" terms absorb the properties of a note that cause it to be liked by some users and not others, and the "helpfulness" term only measures the properties that a note has that cause it to be liked by all. Thus, selecting for helpfulness identifies notes that get cross-tribal approval, and selects against notes that get cheering from one tribe at the expense of disgust from the other tribe.

I made a simplified implementation of the basic algorithm; you can find it [here](#), and are welcome to play around with it.

Now, the above is only a description of the central core of the algorithm. In reality, there are a *lot* of extra mechanisms bolted on top. Fortunately, they are described in the [public documentation](#). These mechanisms include the following:

- The algorithm gets run many times, each time adding some randomly generated extreme "pseudo-votes" to the votes. This means that the algorithm's true output for each note is a range of values, and the final result depends on a "lower confidence bound" taken from this range, which is checked against a threshold of 0.32.
- If many users (especially users with a similar polarity to the note) rate a note "Not Helpful", and furthermore they specify the same "[tag](#)" (eg. "Argumentative or biased language", "Sources do not support note") as the reason for their rating, the helpfulness threshold required for the note to be published increases from 0.4 to 0.5 (this looks small but it's very significant in practice)
- If a note is accepted, the threshold that its helpfulness must drop below to de-accept it is 0.01 points lower than the threshold that a note's helpfulness needed to reach for the note to be originally accepted
- The algorithm gets run *even more times* with multiple models, and this can sometimes promote notes whose original helpfulness score is somewhere between 0.3 and 0.4

All in all, you get some pretty complicated python code that amounts to 6282 lines stretching across 22 files. But it is all [open](#), you can download the [note and rating data](#) and run it yourself, and see if the outputs correspond to what is actually on Twitter at any given moment.

So how does this look in practice?

Probably the single most important idea in this algorithm that distinguishes it from naively taking an average score from people's votes is what I call the "polarity" values. The algorithm documentation calls them f_u and f_n, using f for *factor* because these are the two terms that get multiplied with each other; the more general language is in part because of a desire to eventually make f_u and f_n multi-dimensional.

Polarity is assigned to both users and notes. The link between *user* IDs and the underlying Twitter accounts is intentionally kept hidden, but notes are public. In practice, the polarities generated by the algorithm, at least for the English-language data set, map very closely to the left vs right political spectrum.

Here are some examples of notes that have gotten polarities around -0.8:

Note	Polarity
Anti-trans rhetoric has been amplified by some conservative Colorado lawmakers, including U.S. Rep. Lauren Boebert, who narrowly won re-election in Colorado's GOP-leaning 3rd Congressional District, which does not include Colorado Springs. https://coloradosun.com/2022/11/20/colorado-springs-club-q-lgbtq-trans/	-0.800
President Trump explicitly undermined American faith in election results in the months leading up to the 2020 election. https://www.npr.org/2021/02/08/965342252/timeline-what-trump-told-supporters-for-months-before-they-attacked Enforcing Twitter's Terms of Service is not election interference.	-0.825
The 2020 election was conducted in a free and fair manner. https://www.npr.org/2021/12/23/1065277246/trump-big-lie-jan-6-election	-0.818

Note that I am not cherry-picking here; these are literally the first three rows in the `scored_notes.tsv` spreadsheet generated by the algorithm when I ran it locally that have a polarity score (called `coreNoteFactor1` in the spreadsheet) of less than -0.8.

Now, here are some notes that have gotten polarities around +0.8. It turns out that many of these are either people talking about Brazilian politics in Portuguese or Tesla fans angrily refuting criticism of Tesla, so let me cherry-pick a bit to find a few that are not:

Note	Polarity
As of 2021 data, 64% of "Black or African American" children lived in single-parent families. https://datacenter.aecf.org/data/tables/107-children-in-single-parent-families-by-race-and-ethnicity	+0.809
Contrary to Rolling Stones push to claim child trafficking is "a Qanon adjacent conspiracy," child trafficking is a real and huge issue that this movie accurately depicts. Operation Underground Railroad works with multinational agencies to combat this issue. https://ourrescue.org/	+0.840
Example pages from these LGBTQ+ children's books being banned can be seen here: https://i.imgur.com/8SY6cEx.png These books are obscene, which is not protected by the US constitution as free speech. https://www.justice.gov/criminal-ceos/obscenity "Federal law strictly prohibits the distribution of obscene matter to minors.	+0.806

Once again, it is worth reminding ourselves that the "left vs right divide" was *not* in any way hardcoded into the algorithm; it was discovered emergently by the calculation. This suggests that if you apply this algorithm in other cultural contexts, it could automatically detect what their primary political divides are, and bridge across those too.

Meanwhile, notes that get the highest *helpfulness* look like this. This time, because these notes are actually shown on Twitter, I can just screenshot one directly:

And another one:

The second one touches on highly partisan political themes more directly, but it's a clear, high-quality and informative note, and so it gets rated highly. So all in all, the algorithm seems to work, and the ability to verify the outputs of the algorithm by running the code seems to work.

What do I think of the algorithm?

The main thing that struck me when analyzing the algorithm is just how *complex* it is. There is the "academic paper version", a gradient descent which finds a best fit to a five-term vector and matrix equation, and then the real version, a complicated series of many different executions of the algorithm with lots of arbitrary coefficients along the way.

Even the academic paper version hides complexity under the hood. The equation that it's optimizing is a [degree](#)-4 equation (as there's a degree-2 $f_u * f_n$ term in the prediction formula, and compounding that the cost function [measures error squared](#)). While optimizing a degree-2 equation over any number of variables almost always has a unique solution, which you can calculate with fairly basic linear algebra, a degree-4 equation over many variables often has many solutions, and so multiple rounds of a gradient descent algorithm may well arrive at different answers. Tiny changes to the input may well cause the descent to flip from one local minimum to another, significantly changing the output.

The distinction between this, and algorithms that I helped work on such as [quadratic funding](#), feels to me like a distinction between an **economist's algorithm** and an **engineer's algorithm**. An economist's algorithm, at its best, values being simple, being reasonably easy to analyze, and having clear mathematical properties that show why it's optimal (or least-bad) for the task that it's trying to solve, and ideally proves bounds on how much damage someone can do by trying to exploit it. An engineer's algorithm, on the other hand, is a result of iterative trial and error, seeing what works and what doesn't in the engineer's operational context. Engineer's algorithms *are pragmatic and do the job*; economist's algorithms *don't go totally crazy when confronted with the unexpected*.

Or, as was [famously said](#) on a related topic by the esteemed internet philosopher roon (aka tszzl):

> **roon** ✔
> @tszzl
>
> deep learning vs crypto is a clear divide of rotators vs wordcels. the former offends theorycel aesthetic sensibilities but empirically works to produce absurd miracles. the latter is an insane series of nerd traps and sky high abstraction ladders yet mostly scams
>
> 2:00 PM · Dec 21, 2021

Of course, I would say that the "theorycel aesthetic" side of crypto is necessary precisely to distinguish protocols that are [actually trustless](#) from janky constructions that look fine and seem to work well but under the hood require trusting a few centralized actors - or worse, actually end up being [outright scams](#).

Deep learning works when it works, but it has inevitable vulnerabilities to all kinds of [adversarial machine learning](#) attacks. Nerd traps and sky-high abstraction ladders, if done well, can be quite robust against them. And so one question I have is: **could we turn Community Notes itself into something that's more like an economist algorithm**?

To give a view of what this would mean in practice, let's explore an algorithm I came up with a few years ago for a similar purpose: [pairwise-bounded quadratic funding](#).

Green area: sum of contributions

Yellow area: subsidy if k=1

The goal of pairwise-bounded quadratic funding is to plug a hole in "regular" quadratic funding, where if even two participants collude with each other, they can each contribute a very high amount of money to a fake project that sends the money back to them, and get a large subsidy that drains the entire pool. In pairwise quadratic funding, we assign each pair of participants a limited budget M. The algorithm walks over all possible pairs of participants, and if the algorithm decides to add a subsidy to some project P because both participant A and participant B supported it, that subsidy comes out of the budget assigned to the pair (A, B). Hence, even if k participants were to collude, the amount they could steal from the mechanism is at most $k * (k-1) * M$.

An algorithm of *exactly* this form is not very applicable to the Community Notes context, because each user makes very few votes: on average, any two users would have exactly zero votes in common, and so the algorithm would learn nothing about users' polarities by just looking at each pair of users separately. The goal of the machine learning model is precisely to try to "fill in" the matrix from very sparse source data that cannot be analyzed in this way directly. But the challenge of this approach is that it takes extra effort to do it in a way that does not make the result highly volatile in the face of a few bad votes.

Does Community Notes actually fight polarization?

One thing that we could do is analyze whether or not the Community Notes algorithm, as is, actually manages to fight polarization *at all* - that is, whether or not it actually does any better than a naive voting algorithm. Naive voting algorithms already fight polarization to some limited extent: a post with 200 upvotes and 100 downvotes does worse than a post that just gets the 200 upvotes. But does Community Notes do better than *that*?

Looking at the algorithm abstractly, it's hard to tell. Why wouldn't a high-average-rating but polarizing post get a strong polarity *and* a high helpfulness? The idea is that polarity is supposed to "absorb" the properties of a note that cause it to get a lot of votes if those votes are conflicting, but does it actually do that?

To check this, I ran my own simplified implementation for 100 rounds. The average results were:

```
Quality averages:
Group 1 (good): 0.30032841807271166
Group 2 (good but extra polarizing): 0.21698871680927437
Group 3 (neutral): 0.09443120045416832
Group 4 (bad): -0.1521160965793673
```

In this test, "Good" notes received a rating of +2 from users in the same political tribe and +0 from users in the opposite political tribe, and "Good but extra polarizing" notes received a rating of +4 from same-tribe users and -2 from opposite-tribe users. Same average, but different polarity. And it seems to actually be the case that "Good" notes get a higher average helpfulness than "Good but extra polarizing" notes.

One other benefit of having something closer to an "economist's algorithm" would be having a clearer story for how the algorithm is penalizing polarization.

How useful is this all in high-stakes situations?

We can see some of how this works out by looking at one specific situation. About a month ago, Ian Bremmer complained that a highly critical Community Note that was added to a tweet by a Chinese government official had been removed.

The note, which is now no longer visible. Screenshot by [Ian Bremmer](#).

This is heavy stuff. It's one thing to do mechanism design in a nice sandbox Ethereum community environment where the largest complaint is $20,000 [going to a polarizing Twitter influencer](#). It's another to do it for political and geopolitical questions that affect many millions of people and where everyone, often quite understandably, is assuming maximum bad faith. But if mechanism designers want to have a significant impact into the world, engaging with these high-stakes environments is ultimately necessary.

In the case of Twitter, there is a clear reason why one might suspect centralized manipulation to be behind the Note's removal: Elon has a lot of [business interests in China](#), and so there is a possibility that Elon forced the Community Notes team to interfere with the algorithm's outputs and delete this specific one.

Fortunately, the algorithm is open source and verifiable, so we can actually look under the hood! Let's do that. The URL of the original tweet is https://twitter.com/MFA_China/status/1676157337109946369. The number at the end, 1676157337109946369, is the tweet ID. We can search for that in the [downloadable data](#), and identify the specific row in the spreadsheet that has the above note:

Here we get the ID of the note itself, `1676391378815709184`. We then search for *that* in the `scored_notes.tsv` and `note_status_history.tsv` files generated by running the algorithm. We get:

The second column in the first output is the note's current rating. The second output shows the note's history: its current status is in the seventh column (`NEEDS_MORE_RATINGS`), and the first status that's not `NEEDS_MORE_RATINGS` that it received earlier on is in the fifth column (`CURRENTLY_RATED_HELPFUL`). Hence, we see that **the algorithm itself first showed the note, and then removed it once its rating dropped somewhat - seemingly no centralized intervention involved**.

We can see this another way by looking at the votes themselves. We can scan the `ratings-00000.tsv` file to isolate all the ratings for this note, and see how many rated `HELPFUL` vs `NOT_HELPFUL`:

But if you sort them by timestamp, and look at the first 50 votes, you see 40 `HELPFUL` votes and 9 `NOT_HELPFUL` votes. And so we see the same conclusion: the note's initial audience viewed the note more favorably then the note's later audience, and so its rating started out higher and dropped lower over time.

Unfortunately, the exact story of *how* the note changed status is complicated to explain: it's not a simple matter of "before the rating was above 0.40, now it's below 0.40, so it got dropped". Rather, the high volume of `NOT_HELPFUL` replies triggered one of the [outlier conditions](#), increasing the helpfulness score that the note needs to stay over the threshold.

This is a good learning opportunity for another lesson: making a credibly neutral algorithm truly *credible* requires [keeping it simple](#). If a note moves from being accepted to not being accepted, there should be a simple and legible story as to why.

Of course, there is a totally different way in which this vote could have been manipulated: brigading. Someone who sees a note that they disapprove of could call upon a highly engaged community (or worse, a mass of fake accounts) to rate it `NOT_HELPFUL`, and it may not require that many votes to drop the note from being seen as "helpful" to being seen as "polarized". Properly minimizing the vulnerability of this algorithm to such coordinated attacks will require a lot more analysis and work. One possible improvement would be not allowing any user to vote on any note, but instead using the "For you" algorithmic feed to randomly allocate notes to raters, and only allow raters to rate those notes that they have been allocated to.

Is Community Notes not "brave" enough?

The main criticism of Community Notes that I have seen is basically that it does not do enough. Two recent articles that I have seen make this point. Quoting one:

> The program is severely hampered by the fact that for a Community Note to be public, it has to be generally accepted by a consensus of people from all across the political spectrum.
>
> "It has to have ideological consensus," he said. "That means people on the left and people on the right have to agree that that note must be appended to that tweet."
>
> Essentially, it requires a "cross-ideological agreement on truth, and in an increasingly partisan environment, achieving that consensus is almost impossible, he said.

This is a difficult issue, but ultimately I come down on the side that it is better to let ten misinformative tweets go free than it is to have one tweet covered by a note that judges it unfairly. We have seen years of fact-checking that *is* brave, and *does* come from the perspective of "well, actually we know the truth, and we know that one side lies much more often than the other". And what happened as a result?
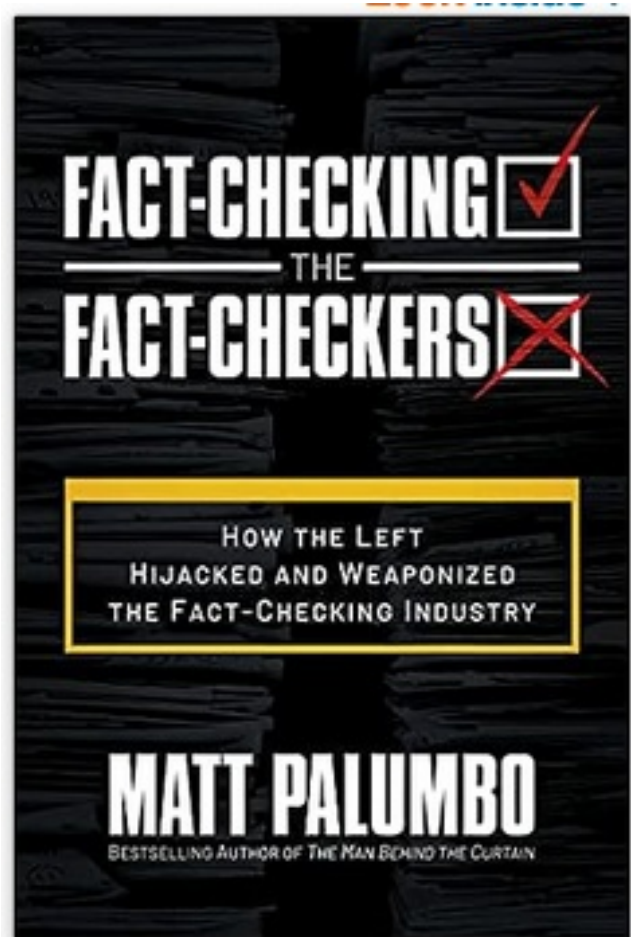
Honestly, some pretty widespread distrust of fact-checking as a concept. One strategy here is to say: ignore the haters, remember that the fact checking experts really do know the facts better than any voting system, and stay the course. But going all-in on this approach seems risky. There is value in building cross-tribal institutions that are at least somewhat respected by everyone. As with William Blackstone's dictum and the courts, it feels to me that maintaining such respect requires a system that commits far more sins of omission than it does sins of commission. And so it seems valuable to me that there is at least one major organization that is taking this alternate path, and treating its rare cross-tribal respect as a resource to be cherished and built upon.

Another reason why I think it is okay for Community Notes to be conservative is that I do not think it is the goal for every misinformative tweet, or even most misinformative tweets, to receive a corrective note. **Even if less than one percent of misinformative tweets get a note providing context or correcting them, Community Notes is still providing an exceedingly valuable service as an educational tool.** The goal is not to correct everything; rather, the goal is to remind people that multiple perspectives exist, that certain kinds of posts that look convincing and engaging in isolation are actually quite incorrect, and you, yes you, can often go do a basic internet search to verify that it's incorrect.

Community Notes cannot be, and is not meant to be, a miracle cure that solves all problems in public epistemology. Whatever problems it does not solve, there is plenty of room for other mechanisms, whether newfangled gadgets such as prediction markets or good old-fashioned organizations hiring full-time staff with domain expertise, to try to fill in the gaps.

Conclusions

Community Notes, in addition to being a fascinating social media experiment, is also an instance of a fascinating new and emerging genre of mechanism design: mechanisms that intentionally try to identify polarization, and favor things that bridge across divides rather than perpetuate them.

The two other things in this category that I know about are (i) pairwise quadratic funding, which is being used in Gitcoin Grants and (ii) Polis, a discussion tool that uses clustering algorithms to help communities identify statements that are commonly well-received across people who normally have different viewpoints. This area of mechanism design is valuable, and I hope that we can see a lot more academic work in this field.

Algorithmic transparency of the type that Community Notes offers is not quite full-on decentralized social media - if you disagree with how Community Notes works, there's no way to go see a view of the same content with a different algorithm. But it's the closest that very-large-scale applications are going to get within the next couple of years, and we can see that it provides a lot of value already, both by preventing centralized manipulation and by ensuring that platforms that do not engage in such manipulation can get proper credit for doing so.

I look forward to seeing both Community Notes, and hopefully many more algorithms of a similar spirit, develop and grow over the next decade.

What do I think about biometric proof of personhood?

2023 Jul 24

Special thanks to the Worldcoin team, the Proof of Humanity community and Andrew Miller for discussion.

One of the trickier, but potentially one of the most valuable, gadgets that people in the Ethereum community have been trying to build is a decentralized proof-of-personhood solution. Proof of personhood, aka the "unique-human problem", is a limited form of real-world identity that asserts that a given registered account is controlled by a real person (and a different real person from every other registered account), ideally without revealing *which* real person it is.

There have been a few efforts at tackling this problem: Proof of Humanity, BrightID, Idena and Circles come up as examples. Some of them come with their own applications (often a UBI token), and some have found use in Gitcoin Passport to verify which accounts are valid for quadratic voting. Zero-knowledge tech like Sismo adds privacy to many of these solutions. More recently, we have seen the rise of a much larger and more ambitious proof-of-personhood project: Worldcoin.

Worldcoin was co-founded by Sam Altman, who is best known for being the CEO of OpenAI. The philosophy behind the project is simple: AI is going to create a lot of abundance and wealth for humanity, but it also may kill very many people's jobs and make it almost impossible to tell who even is a human and not a bot, and so we need to plug that hole by (i) creating a really good proof-of-personhood system so that humans can prove that they actually are humans, and (ii) giving everyone a UBI. Worldcoin is unique in that it relies on highly sophisticated biometrics, scanning each user's iris using a piece of specialized hardware called "the Orb":

The goal is to produce a large number of these Orbs and widely distribute them around the world and put them in public places to make it easy for anyone to get an ID. To Worldcoin's credit, they have also committed to decentralize over time. At first, this means technical decentralization: being an L2 on Ethereum using the Optimism stack, and protecting users' privacy with ZK-SNARKs and other cryptographic techniques. Later on, it includes decentralizing governance of the system itself.

Worldcoin has been criticized for privacy and security concerns around the Orb, design issues in its "coin", and for ethical issues around some choices that the company has made. Some of the criticisms are highly specific, focusing on decisions made by the project that could easily have been made in another way - and indeed, that the Worldcoin project itself may be willing to change. Others, however, raise the more fundamental concern of whether or not biometrics - not just the eye-scanning biometrics of Worldcoin, but also the simpler face-video-uploads and verification games used in Proof of Humanity and Idena - are a good idea at all. And still others criticize proof of personhood in general. Risks include unavoidable privacy leaks, further erosion of people's ability to navigate the internet anonymously, coercion by authoritarian governments, and the potential impossibility of being secure at the same time as being decentralized.
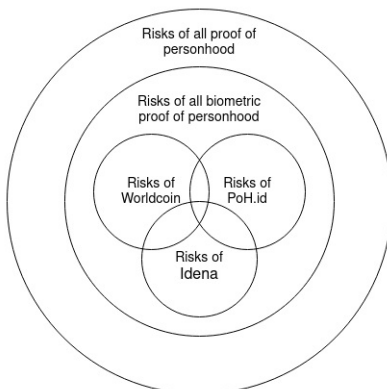
This post will talk about these issues, and go through some arguments that can help you decide whether or not bowing down and scanning your eyes (or face, or voice, or...) before our new spherical overlords is a good idea, and whether or not the natural alternatives - either using social-graph-based proof of personhood or giving up on proof of personhood entirely - are any better.

What is proof of personhood and why is it important?

The simplest way to define a proof-of-personhood system is: it creates a list of public keys where the system guarantees that each key is controlled by a unique human. In other words, if you're a human, you can put one key on the list, but you can't put two keys on the list, and if you're a bot you can't put any keys on the list.

Proof of personhood is valuable because it solves a lot of anti-spam and anti-concentration-of-power problems that many people have, in a way that avoids dependence on centralized authorities and reveals the minimal information possible. If proof of personhood is not solved, decentralized governance (including "micro-governance" like votes on social media posts) becomes much easier to capture by very wealthy actors, including hostile governments. Many services would only be able to prevent denial-of-service attacks by setting a price for access, and sometimes a price high enough to keep out attackers is also too high for many lower-income legitimate users.

Many major applications in the world today deal with this issue by using government-backed identity systems such as credit cards and passports. This solves the problem, but it makes large and perhaps unacceptable sacrifices on privacy, and can be trivially attacked by governments themselves.

How many proof of personhood proponents see the two-sided risk that we are facing. Image source.

In many proof-of-personhood projects - not just Worldcoin, but also Proof of Humanity, Circles and others - the "flagship application" is a built-in "N-per-person token" (sometimes called a "UBI token"). Each user registered in the system receives some fixed quantity of tokens each day (or hour, or week). But there are plenty of other applications:

- Airdrops for token distributions
- Token or NFT sales that give more favorable terms to less-wealthy users
- Voting in DAOs
- A way to "seed" graph-based reputation systems
- Quadratic voting (and funding, and attention payments)
- Protection against bots / sybil attacks in social media
- An alternative to captchas for preventing DoS attacks

In many of these cases, the common thread is a desire to create mechanisms that are open and democratic, avoiding both centralized control by a project's operators and domination by its wealthiest users. The latter is especially important in decentralized governance. In many of these cases, existing solutions today rely on some combination of (i) highly opaque AI algorithms that leave lots of room to undetectably discriminate against users that the operators simply do not like, and (ii) centralized IDs, aka "KYC". An effective proof-of-personhood solution would be a much better alternative, achieving the security properties that those applications need without the pitfalls of the existing centralized approaches.

What are some early attempts at proof of personhood?

There are two main forms of proof of personhood: **social-graph-based** and **biometric**. Social-graph based proof of personhood relies on some form of vouching: if Alice, Bob, Charlie and David are all verified humans, and they all say that Emily is a verified human, then Emily is probably also a verified human. Vouching is often enhanced with incentives: if Alice says that Emily is a human, but it turns out that she is not, then Alice and Emily may both get penalized. Biometric proof of personhood involves verifying some physical or behavioral trait of Emily, that distinguishes humans from bots (and individual humans from each other). Most projects use a combination of the two techniques.

The four systems I mentioned at the beginning of the post work roughly as follows:

- **Proof of Humanity**: you upload a video of yourself, and provide a deposit. To be approved, an existing user needs to vouch for you, and an amount of time needs to pass during which you can be challenged. If there is a challenge, a Kleros decentralized court determines whether or not your video was genuine; if it is not, you lose your deposit and the challenger gets a reward.
- **BrightID**: you join a video call "verification party" with other users, where everyone verifies each other. Higher levels of verification are available via Bitu, a system in which you can get verified if enough other Bitu-verified users vouch for you.
- **Idena**: you play a captcha game at a specific point in time (to prevent people from participating multiple times); part of the captcha game involves creating and verifying captchas that will then be used to verify others.
- **Circles**: an existing Circles user vouches for you. Circles is unique in that it does not attempt to create a "globally verifiable ID"; rather, it creates a graph of trust relationships, where someone's trustworthiness can only be verified from the perspective of your own position in that graph.

How does Worldcoin work?

Each Worldcoin user installs an app on their phone, which generates a private and public key, much like an Ethereum wallet. They then go in-person to visit an "Orb". The user stares into the Orb's camera, and at the same time shows the Orb a QR code generated by their Worldcoin app, which contains their public key. The Orb scans the user's eyes, and uses complicated hardware scanning and machine-learned classifiers to verify that:

1. The user is a real human
2. The user's iris does not match the iris of any other user that has previously used the system

If both scans pass, the Orb signs a message approving a specialized hash of the user's iris scan. The hash gets uploaded to a database - currently a centralized server, intended to be replaced with a decentralized on-chain system once they are sure the hashing mechanism works. The system does not store full iris scans; it only stores hashes, and these hashes are used to check for uniqueness. From that point forward, the user has a "World ID".

A World ID holder is able to prove that they are a unique human by generating a ZK-SNARK proving that they hold the private key corresponding to a public key in the database, without revealing *which* key they hold. Hence, even if someone re-scans your iris, they will not be able to see any actions that you have taken.

What are the major issues with Worldcoin's construction?

There are four major risks that immediately come to mind:

- **Privacy**. The registry of iris scans may reveal information. At the very least, if *someone else* scans your iris, they can check it against the database to determine whether or not you have a World ID. Potentially, iris scans might reveal more information.
- **Accessibility**. World IDs are not going to be reliably accessible unless there are so many Orbs that anyone in the world can easily get to one.
- **Centralization**. The Orb is a hardware device, and we have no way to verify that it was constructed correctly and does not have backdoors. Hence, even if the software layer is perfect and fully decentralized, the Worldcoin Foundation still has the ability to insert a backdoor into the system, letting it create arbitrarily many fake human identities.
- **Security**. Users' phones could be hacked, users could be coerced into scanning their irises while showing a public key that belongs to someone else, and there is the possibility of 3D-printing "fake people" that can pass the iris scan and get World IDs.

It's important to distinguish between (i) issues specific to choices made by Worldcoin, (ii) issues that *any biometric proof of personhood* will inevitably have, and (iii) issues that *any proof of personhood in general* will have. For example, signing up to Proof of Humanity means publishing your face on the internet. Joining a BrightID verification party doesn't *quite* do that, but still exposes who you are to a lot of people. And joining Circles publicly exposes your social graph. Worldcoin is significantly *better* at preserving privacy than either of those. On the other hand, Worldcoin depends on specialized hardware, which opens up the challenge of trusting the orb manufacturers to have constructed the orbs correctly - a challenge which has no parallels in Proof of Humanity, BrightID or Circles. It's even conceivable that in the future, someone other than Worldcoin will create a *different* specialized-hardware solution that has different tradeoffs.

How do biometric proof-of-personhood schemes address privacy issues?

The most obvious, and greatest, potential privacy leak that any proof-of-personhood system has is linking each action that a person takes to a real-world identity. This data leak is very large, arguably unacceptably large, but fortunately it is easy to solve with **zero knowledge proof** technology. Instead of directly making a signature with a private key whose corresponding public key is in the database, a user could make a ZK-SNARK proving that they own the private key whose corresponding public key is *somewhere* in the database, without revealing which specific key they have. This can be done generically with tools like Sismo (see here for the Proof of Humanity-specific implementation), and Worldcoin has its own built-in implementation. **It's important to give "crypto-native" proof of personhood credit here: they actually care about taking this basic step to provide anonymization, whereas basically all centralized identity solutions do not.**

A more subtle, but still important, privacy leak is the mere *existence* of a public registry of biometric scans. In the case of Proof of Humanity, this is a lot of data: you get a video of each Proof of Humanity participant, making it very clear to anyone in the world who cares to investigate who all the Proof of Humanity participants are. In the case of Worldcoin, the leak is much more limited: the Orb locally computes and publishes only a "hash" of each person's iris scan. This hash is not a regular hash like SHA256; rather, it is a specialized algorithm based on machine-learned Gabor filters that deals with the inexactness inherent in any biometric scan, and ensures that successive hashes taken of the same person's iris have similar outputs.
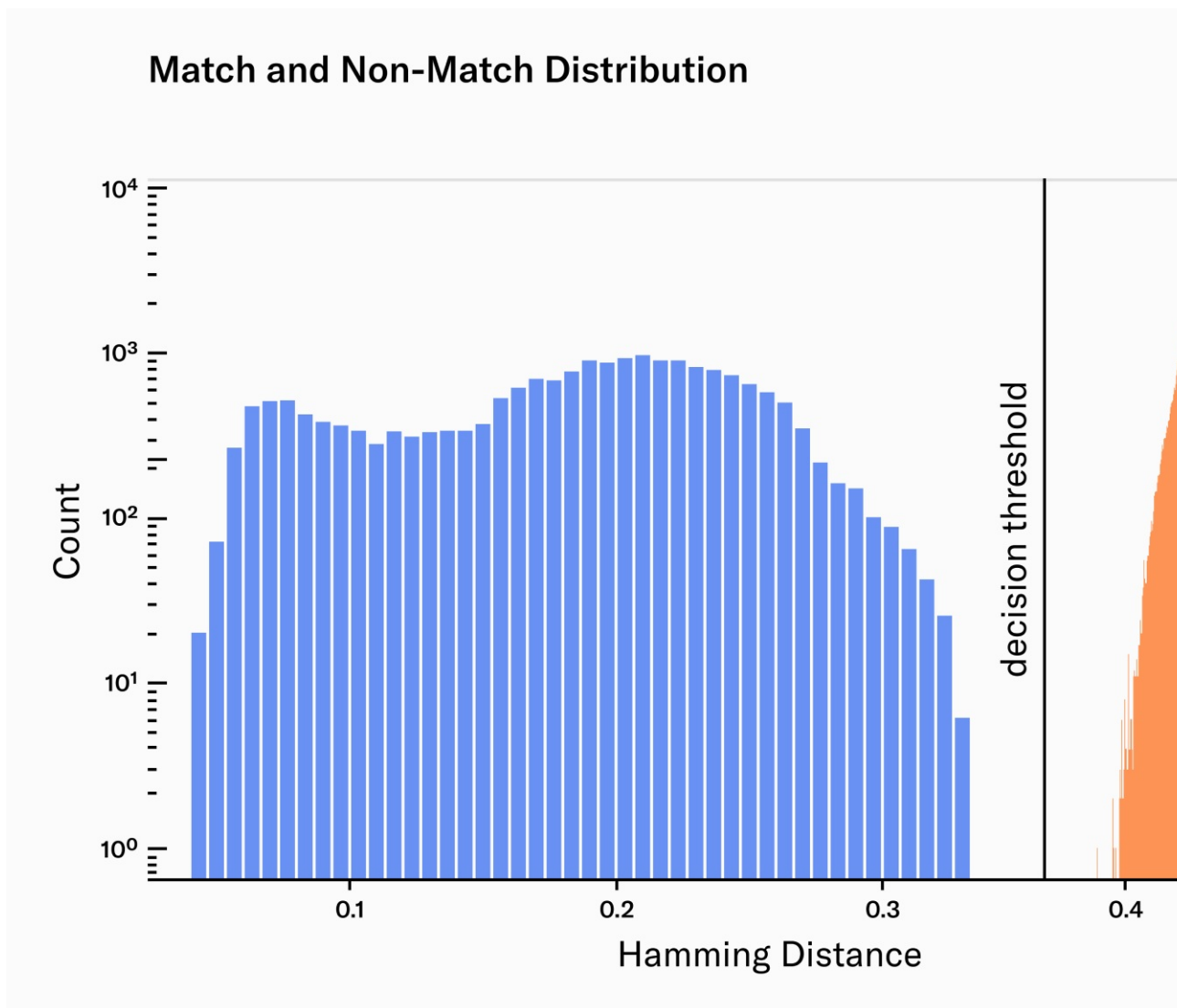
Blue: percent of bits that differ between two scans of the same person's iris. Orange: percent of bits that differ between two scans of two different people's irises.

These iris hashes leak only a small amount of data. If an adversary can forcibly (or secretly) scan your iris, then they can compute your iris hash themselves, and check it against the database of iris hashes to see whether or not you participated in the system. This ability to check whether or not someone signed up is necessary for the system itself to prevent people from signing up multiple times, but there's always the possibility that it will somehow be abused. Additionally, there is the possibility that the iris hashes leak some amount of medical data (sex, ethnicity, perhaps medical conditions), but this leak is far smaller than what could be captured by pretty much any other mass data-gathering system in use today (eg. even street cameras). On the whole, to me the privacy of storing iris hashes seems sufficient.

If others disagree with this judgement and decide that they want to design a system with even more privacy, there are two ways to do so:

1. If the iris hashing algorithm can be improved to make the difference between two scans of the same person much lower (eg. reliably under 10% bit flips), then instead of storing full iris hashes, the system can store a smaller number of error correction bits for iris hashes (see: fuzzy extractors). If the difference between two scans is under 10%, then the number of bits that needs to be published would be at least 5x less.
2. If we want to go further, we could store the iris hash database inside a multi-party computation (MPC) system which could only be accessed by Orbs (with a rate limit), making the data inaccessible entirely, but at the cost of significant protocol complexity and social complexity in governing the set of MPC participants. This would have the benefit that users would not be able to prove a link between two different World IDs that they had at different times even if they wanted to.

Unfortunately, these techniques are not applicable to Proof of Humanity, because Proof of Humanity requires the full video of each participant to be publicly available so that it can be challenged if there are signs that it is fake (including AI-generated fakes), and in such cases investigated in more detail.

On the whole, despite the "dystopian vibez" of staring into an Orb and letting it scan deeply into your eyeballs, it does seem like specialized hardware systems can do quite a decent job of protecting privacy. However, the flip side of this is that specialized hardware systems introduce much greater centralization concerns. Hence, we cypherpunks seem to be stuck in a bind: we have to trade off one deeply-held cypherpunk value against another.

What are the accessibility issues in biometric proof-of-personhood systems?

Specialized hardware introduces accessibility concerns because, well, specialized hardware is not very accessible. Somewhere between [51%](#) and [64%](#) of sub-Saharan Africans now have smartphones, and this seems to be projected to [increase to 87%](#) by 2030. But while there are billions of smartphones, there are only a few hundred Orbs. Even with much higher-scale distributed manufacturing, it would be hard to get to a world where there's an Orb within five kilometers of *everyone*.

But to the team's credit, [they have been trying](#)!

It is also worth noting that many *other* forms of proof of personhood have accessibility problems that are even worse. It is very difficult to join a social-graph-based proof-of-personhood system unless you already know someone who is in the social graph. This makes it very easy for such systems to remain restricted to a single community in a single country.

Even *centralized* identity systems have learned this lesson: India's [Aadhaar ID system](#) is biometric-based, as that was the only way to quickly onboard its [massive population](#) while avoiding massive fraud from duplicate and fake accounts (resulting in [huge cost savings](#)), though of course the Aadhaar system as a whole is far weaker on privacy than anything being proposed on a large scale within the crypto community.

The best-performing systems from an accessibility perspective are actually systems like Proof of Humanity, which you can sign up to using only a smartphone - though, as we have seen and as we will see, such systems come with all kinds of other tradeoffs.

What are the centralization issues in biometric proof-of-personhood systems?

There are three:

1. Centralization risks in the system's top-level governance (esp. the system that makes final top-level resolutions if different actors in the system disagree on subjective judgements).
2. Centralization risks unique to systems that use specialized hardware.
3. Centralization risks if proprietary algorithms are used to determine who is an authentic participant.

Any proof-of-personhood system must contend with (1), perhaps with the exception of systems where the set of "accepted" IDs is completely subjective. If a system uses incentives denominated in outside assets (eg. ETH, USDC, DAI), then it cannot be fully subjective, and so governance risks become unavoidable.

[2] is a much bigger risk for Worldcoin than for Proof of Humanity (or BrightID), because Worldcoin depends on specialized hardware and other systems do not.

[3] is a risk particularly in "[logically centralized](#)" systems where there is a single system doing the verification, unless all of the algorithms are open-source and we have an assurance that they are actually running the code that they claim they are. For systems that rely purely on users verifying other users (like Proof of Humanity), it is not a risk.

How does Worldcoin address hardware centralization issues?

Currently, a Worldcoin-affiliated entity called [Tools for Humanity](#) is the only organization that is making Orbs. However, the Orb's source code is [mostly public](#): you can see the hardware specs in [this github repository](#), and other parts of the source code are expected to be published soon. The [license](#) is another one of those "shared source but not technically open source until four years from now" licenses similar to the [Uniswap BSL](#), except in addition to preventing forking it also prevents what they consider unethical behavior - they specifically list mass surveillance and [three](#) international [civil](#) rights [declarations](#).

The team's stated goal is to allow and encourage other organizations to create Orbs, and over time transition from Orbs being created by Tools for Humanity to having some kind of DAO that approves and manages which organizations can make Orbs that are recognized by the system.

There are two ways in which this design can fail:

1. **It fails to actually decentralize**. This could happen because of the common [trap of federated protocols](#): one manufacturer ends up dominating in practice, causing the system to re-centralize. Presumably, governance could limit how many valid Orbs each manufacturer can produce, but this would need to be managed carefully, and it puts a lot of pressure on governance to be both decentralized *and* monitor the ecosystem and respond to threats effectively: a much harder task than eg. a fairly static DAO that just handles top-level dispute resolution tasks.
2. **It turns out that it's not possible to make such a distributed manufacturing mechanism secure**. Here, there are two risks that I see:
   - **Fragility against bad Orb manufacturers**: if even one Orb manufacturer is malicious or hacked, it can generate an unlimited number of fake iris scan hashes, and give them World IDs.
   - **Government restriction of Orbs**: governments that do not want their citizens participating in the Worldcoin ecosystem can ban Orbs from their country. Furthermore, they could even force their citizens to get their irises scanned, allowing the government to get their accounts, and the citizens would have no way to respond.

To make the system more robust against bad Orb manufacturers, the Worldcoin team is proposing to perform regular audits on Orbs, verifying that they are built correctly and key hardware components were built according to specs and were not tampered with after the fact. This is a challenging task: it's basically something like the [IAEA nuclear inspections bureaucracy](#) but for Orbs. The hope is that even a very imperfect implementation of an auditing regime could greatly cut down on the number of fake Orbs.

To limit the harm caused by any bad Orb that *does* slip through, it makes sense to have a second mitigation. **World IDs registered with different Orb manufacturers, and ideally with different Orbs, should be distinguishable from each other**. It's okay if this information is private and only stored on the World ID holder's device; but it does need to be provable on demand. This makes it possible for the ecosystem to respond to (inevitable) attacks by removing individual Orb manufacturers, and perhaps even individual Orbs, from the whitelist on-demand. If we see the North Korea government going around and forcing people to scan their eyeballs, those Orbs and any accounts produced by them could be immediately retroactively disabled.

Security issues in proof of personhood in general

In addition to issues specific to Worldcoin, there are concerns that affect proof-of-personhood designs in general. The major ones that I can think of are:

1. **3D-printed fake people**: one could use AI to generate photographs or even 3D prints of fake people that are convincing enough to get accepted by the Orb software. If even one group does this, they can generate an unlimited number of identities.
2. **Possibility of selling IDs**: someone can provide someone else's public key instead of their own when registering, giving that person control of their

registered ID, in exchange for money. This [seems to be happening already](#). In addition to selling, there's also the possibility of **renting IDs** to use for a short time in one application.
3. **Phone hacking**: if a person's phone gets hacked, the hacker can steal the key that controls their World ID.
4. **Government coercion to steal IDs**: a government could force their citizens to get verified while showing a QR code belonging to the government. In this way, a malicious government could gain access to millions of IDs. In a biometric system, this could even be done covertly: governments could use obfuscated Orbs to extract World IDs from everyone entering their country at the passport control booth.

[1] is specific to biometric proof-of-personhood systems. [2] and [3] are common to both biometric and non-biometric designs. [4] is also common to both, though the techniques that are required would be quite different in both cases; in this section I will focus on the issues in the biometric case.

These are pretty serious weaknesses. Some already have been addressed in existing protocols, others can be addressed with future improvements, and still others seem to be fundamental limitations.

How can we deal with fake people?

This is significantly less of a risk for Worldcoin than it is for Proof of Humanity-like systems: an in-person scan can examine many features of a person, and is quite hard to fake, compared to merely [deep-faking](#) a [video](#). Specialized hardware is inherently harder to fool than commodity hardware, which is in turn harder to fool than digital algorithms verifying pictures and videos that are sent remotely.

Could someone 3D-print something that can fool even specialized hardware eventually? Probably. I expect that at some point we will see growing tensions between the goal of keeping the mechanism open and keeping it secure: open-source AI algorithms are inherently more vulnerable to [adversarial machine learning](#). Black-box algorithms are more protected, but it's hard to tell that a black-box algorithm was not trained to include backdoors. Perhaps [ZK-ML technologies](#) could give us the best of both worlds. Though at some point in the even further future, it is likely that even the best AI algorithms will be fooled by the best 3D-printed fake people.

However, from my discussions with both the Worldcoin and Proof of Humanity teams, it seems like at the present moment neither protocol is yet seeing significant deep fake attacks, for the simple reason that **hiring real low-wage workers to sign up on your behalf is quite cheap and easy**.

Can we prevent selling IDs?

In the short term, preventing this kind of outsourcing is difficult, because most people in the world are not even aware of proof-of-personhood protocols, and if you tell them to hold up a QR code and scan their eyes for $30 they will do that. Once more people *are* aware of what proof-of-personhood protocols are, a fairly simple mitigation becomes possible: **allowing people who have a registered ID to re-register, canceling the previous ID**. This makes "ID selling" much less credible, because someone who sells you their ID can just go and re-register, canceling the ID that they just sold. However, getting to this point requires the protocol to be *very* widely known, and Orbs to be *very* widely accessible to make on-demand registration practical.

This is one of the reasons why having a UBI coin integrated into a proof-of-personhood system is valuable: **a UBI coin provides an easily understandable incentive for people to (i) learn about the protocol and sign up, and (ii) immediately re-register if they register on behalf of someone else**. Re-registration also prevents phone hacking.

Can we prevent coercion in biometric proof-of-personhood systems?

This depends on what kind of coercion we are talking about. Possible forms of coercion include:

- Governments scanning people's eyes (or faces, or...) at border control and other routine government checkpoints, and using this to register (and frequently re-register) their citizens
- Governments banning Orbs within the country to prevent people from independently re-registering
- Individuals buying IDs and then threatening to harm the seller if they detect that the ID has been invalidated due to re-registration
- (Possibly government-run) applications requiring people to "sign in" by signing with their public key directly, letting them see the corresponding biometric scan, and hence the link between the user's current ID and any future IDs they get from re-registering. A common fear is that this makes it too easy to create "permanent records" that stick with a person for their entire life.

All your UBI and voting power are belong to us. [Image source](#).

Especially in the hands of unsophisticated users, **it seems quite tough to outright prevent these situations**. Users could leave their country to (re-)register at an Orb in a safer country, but this is a difficult process and high cost. In a truly hostile legal environment, seeking out an independent Orb seems too difficult and risky.

What *is* feasible is making this kind of abuse more *annoying to implement* and *detectable*. The Proof of Humanity approach of requiring a person to speak a specific phrase when registering is a good example: it may be enough to prevent *hidden* scanning, requiring coercion to be much more blatant, and the registration phrase could even include a statement confirming that the respondent knows that they have the right to re-register independently and may get UBI coin or other rewards. If coercion is detected, the devices used to perform coercive registrations en masse could have their access rights revoked. To prevent applications linking people's current and previous IDs and attempting to leave "permanent records", the default proof of personhood app could lock the user's key in trusted hardware, preventing any application from using the key directly without the anonymizing ZK-SNARK layer in between. If a government or application developer wants to get around this, they would need to mandate the use of their own custom app.

With a combination of these techniques and active vigilance, locking out those regimes that are truly hostile, and keeping honest those regimes that are merely medium-bad (as much of the world is), seems possible. This can be done either by a project like Worldcoin or Proof of Humanity maintaining its own bureaucracy for this task, or by revealing more information about how an ID was registered (eg. in Worldcoin, which Orb it came from), and leaving this classification task to the community.

Can we prevent *renting* IDs (eg. to sell votes)?

Renting out your ID is not prevented by re-registration. This is okay in some applications: the cost of renting out your right to collect the day's share of UBI coin is going to be just the value of the day's share of UBI coin. But in applications such as voting, [easy vote selling](#) is a huge problem.

Systems like [MACI](#) can prevent you from credibly selling your vote, by allowing you to later cast another vote that invalidates your previous vote, in such a way that no one can tell whether or not you in fact cast such a vote. However, if the briber controls which key you get *at registration time*, this does not help.

I see two solutions here:

1. **Run entire applications inside an MPC**. This would also cover the re-registration process: when a person registers to the MPC, the MPC assigns them an ID that is separate from, and not linkable to, their proof of personhood ID, and when a person re-registers, only the MPC would know which account to deactivate. This prevents users from making proofs about their actions, because every important step is done inside an MPC using private information that is only known to the MPC.
2. **Decentralized registration ceremonies**. Basically, implement something like [this in-person key-registration protocol](#) that requires four randomly selected local participants to work together to register someone. This could ensure that registration is a "trusted" procedure that an attacker cannot snoop in during.

Social-graph-based systems may actually perform better here, because they can create local decentralized registration processes automatically as a byproduct of how they work.

How do biometrics compare with the other leading candidate for proof of personhood, social graph-based verification?

Aside from biometric approaches, the main other contender for proof of personhood so far has been social-graph-based verification. Social-graph-based verification systems all operate on the same principle: if there are a whole bunch of existing verified identities that all attest to the validity of your identity, then you probably are valid and should also get verified status.
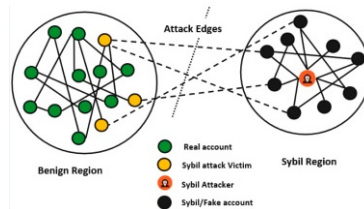
If only a few real users (accidentally or maliciously) verify fake users, then you can use basic graph-theory techniques to put an upper bound on how many fake users get verified by the system. Source: https://www.sciencedirect.com/science/article/abs/pii/S0045790622000611.

Proponents of social-graph-based verification often describe it as being a better alternative to biometrics for a few reasons:

- It **does not rely on special-purpose hardware**, making it much easier to deploy
- It **avoids a permanent arms race** between manufacturers trying to create fake people and the Orb needing to be updated to reject such fake people
- It **does not require collecting biometric data**, making it more privacy-friendly
- It is **potentially more friendly to pseudonymity**, because if someone chooses to split their internet life across multiple identities that they keep separate from each other, both of those identities could potentially be verified (but maintaining multiple genuine and separate identities sacrifices network effects and has a high cost, so it's not something that attackers could do easily)
- **Biometric approaches give a binary score** of "is a human" or "is not a human", which is fragile: people who are accidentally rejected would end up with no UBI at all, and potentially no ability to participate in online life. **Social-graph-based approaches can give a more nuanced numerical score**, which may of course be moderately unfair to some participants but is unlikely to "un-person" someone completely.

My perspective on these arguments is that I largely agree with them! These are genuine advantages of social-graph-based approaches and should be taken seriously. However, it's worth also taking into account the weaknesses of social-graph-based approaches:

- **Bootstrapping**: for a user to join a social-graph-based system, that user must know someone who is already in the graph. This makes large-scale adoption difficult, and risks excluding entire regions of the world that do not get lucky in the initial bootstrapping process.
- **Privacy**: while social-graph-based approaches avoid collecting biometric data, they often end up leaking info about a person's social relationships, which may lead to even greater risks. Of course, zero-knowledge technology can mitigate this (eg. see [this proposal by Barry Whitehat](#)), but the interdependency inherent in a graph and the need to perform mathematical analyses on the graph makes it harder to achieve the same level of data-hiding that you can with biometrics.
- **Inequality**: each person can only have one biometric ID, but a wealthy and socially well-connected person could use their connections to generate many IDs. Essentially, the same flexibility that might allow a social-graph-based system to give multiple pseudonyms to someone (eg. an activist) that really needs that feature would likely also imply that more powerful and well-connected people can gain more pseudonyms than less powerful and well-connected people.
- **Risk of collapse into centralization**: most people are too lazy to spend time reporting into an internet app who is a real person and who is not. As a result, there is a risk that the system will come over time to favor "easy" ways to get inducted that depend on centralized authorities, and the "social graph" that the system users will de-facto become the social graph of which countries recognize which people as citizens - giving us centralized KYC with needless extra steps.

Is proof of personhood compatible with pseudonymity in the real world?

In principle, proof of personhood is compatible with all kinds of pseudonymity. Applications could be designed in such a way that someone with a single proof of personhood ID can create up to five profiles within the application, leaving room for pseudonymous accounts. One could even use [quadratic formulas](#): N accounts for a cost of $N^2. But will they?

A pessimist, however, might argue that it is naive to try to create a more privacy-friendly form of ID and hope that it will actually get adopted in the right way, because the powers-that-be are not privacy-friendly, and if a powerful actor gets a tool that *could* be used to get much more information about a person, they *will* use it that way. In such a world, the argument goes, the only *realistic* approach is, unfortunately, to throw sand in the gears of *any* identity solution, and defend a world with full anonymity and digital islands of high-trust communities.

I see the reasoning behind this way of thinking, but I worry that such an approach would, even if successful, lead to a world where there's no way for anyone to do anything to counteract wealth concentration and governance centralization, because one person could always pretend to be ten thousand. Such points of centralization would, in turn, be easy for the powers-that-be to capture. Rather, I would favor a moderate approach, where we vigorously advocate for proof-of-personhood solutions to have strong privacy, potentially if desired even include a "N accounts for $N^2" mechanism at protocol layer, and create something that has privacy-friendly values *and* has a chance of getting accepted by the outside world.

So... what do I think?

There is no ideal form of proof of personhood. Instead, we have at least three different paradigms of approaches that all have their own unique strengths and weaknesses. A comparison chart might look as follows:

	Social-graph-based	General-hardware biometric	Specialized-hardware biometric
Privacy	Low	Fairly low	Fairly high
Accessibility / scalability	Fairly low	High	Medium
Robustness of decentralization	Fairly high	Fairly high	Fairly low
Security against "fake people"	High (if done well)	Low	Medium

What we should ideally do is treat these three techniques as complementary, and combine them all. As India's Aadhaar has shown at scale, specialized-hardware biometrics have their benefits of being secure at scale. They are very weak at decentralization, though this can be addressed by holding individual Orbs accountable. General-purpose biometrics can be adopted very easily today, but their security is rapidly dwindling, and they may only work for another 1-2 years. Social-graph-based systems bootstrapped off of a few hundred people who are socially close to the founding team are likely to face constant tradeoffs between completely missing large parts of the world and being vulnerable to attacks within communities they have no visibility into. A social-graph-based system bootstrapped off tens of millions of biometric ID holders, however, could actually work. Biometric bootstrapping may work better short-term, and social-graph-based techniques may be more robust long-term, and take on a larger share of the responsibility over time as their algorithms improve.
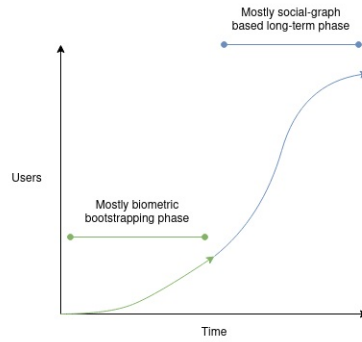
A possible hybrid path.

All of these teams are in a position to make many mistakes, and there are inevitable tensions between business interests and the needs of the wider community, so it's important to exercise a lot of vigilance. As a community, we can and should push all participants' comfort zones on open-sourcing their tech, demand third-party audits and even third-party-written software, and other checks and balances. We also need more alternatives in each of the three categories.

At the same time it's important to recognize the work already done: many of the teams running these systems have shown a willingness to take privacy much more seriously than pretty much any government or major corporate-run identity systems, and this is a success that we should build on.

The problem of making a proof-of-personhood system that is effective and reliable, especially in the hands of people distant from the existing crypto community, seems quite challenging. I definitely do not envy the people attempting the task, and it will likely take years to find a formula that works. The concept of proof-of-personhood in principle seems very valuable, and while the various implementations have their risks, not having any proof-of-personhood at all has its risks too: a world with no proof-of-personhood seems more likely to be a world dominated by centralized identity solutions, money, small closed communities, or some combination of all three. I look forward to seeing more progress on all types of proof of personhood, and hopefully seeing the different approaches eventually come together into a coherent whole.

Deeper dive on cross-L2 reading for wallets and other use cases

2023 Jun 20

Special thanks to Yoav Weiss, Dan Finlay, Martin Koppelmann, and the Arbitrum, Optimism, Polygon, Scroll and SoulWallet teams for feedback and review.

In [this post on the Three Transitions](#), I outlined some key reasons why it's valuable to start thinking explicitly about L1 + cross-L2 support, wallet security, and privacy as necessary basic features of the ecosystem stack, rather than building each of these things as addons that can be designed separately by individual wallets.

This post will focus more directly on the technical aspects of one specific sub-problem: how to make it easier to read L1 from L2, L2 from L1, or an L2 from another L2. Solving this problem is crucial for implementing an asset / keystore separation architecture, but it also has valuable use cases in other areas, most notably optimizing reliable cross-L2 calls, including use cases like moving assets between L1 and L2s.

Recommended pre-reads

- Post on [the Three Transitions](#)
- Ideas from the Safe team on [holding assets across multiple chains](#)
- Why we need wide adoption of [social recovery wallets](#)
- [ZK-SNARKs](#), and [some privacy applications](#)
- [Dankrad on KZG commitments](#)
- [Verkle trees](#)

Table of contents

What is the goal?

Once L2s become more mainstream, users will have assets across multiple L2s, and possibly L1 as well. Once smart contract wallets (multisig, social recovery or otherwise) become mainstream, **the keys needed to access some account are going to change over time, and old keys would need to no longer be valid**. Once *both* of these things happen, a user will need to have a way to change the keys that have authority to access many accounts which live in many different places, without making an extremely high number of transactions.

Particularly, we need a way to handle **counterfactual addresses: addresses that have not yet been "registered" in any way on-chain, but which nevertheless need to receive and securely hold funds**. We all depend on counterfactual addresses: when you use Ethereum for the first time, you are able to generate an ETH address that someone can use to pay you, without "registering" the address on-chain (which would require paying txfees, and hence already holding some ETH).

With [EOAs](#), all addresses start off as counterfactual addresses. With smart contract wallets, counterfactual addresses are still possible, largely thanks to [CREATE2](#), which allows you to have an ETH address that can only be filled by a smart contract that has code matching a particular hash.
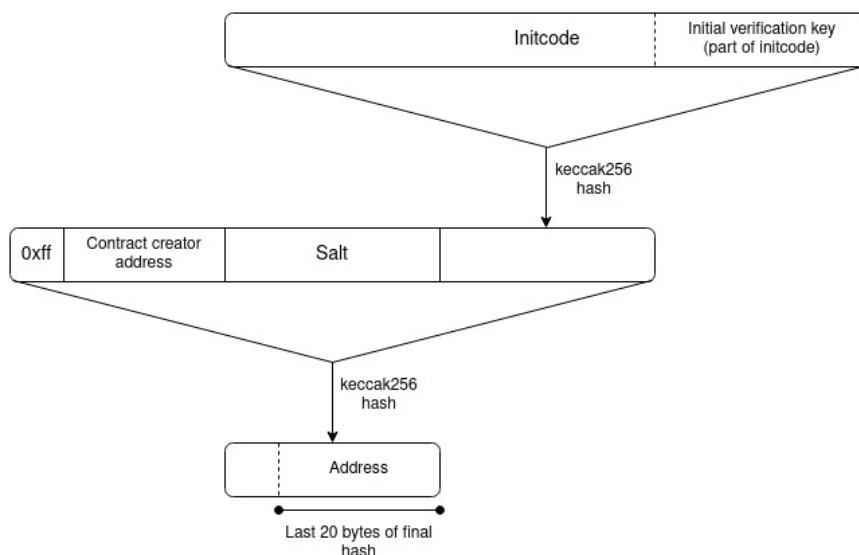
[EIP-1014 (CREATE2)](#) *address calculation algorithm.*

However, smart contract wallets introduce a new challenge: the possibility of access keys *changing*. The address, which is a hash of the `initcode`, can only contain the wallet's *initial* verification key. The *current* verification key would be stored in the wallet's storage, but that storage record does not magically propagate to other L2s.

If a user has many addresses on many L2s, including addresses that (because they are counterfactual) the L2 that they are on does not know about, then it seems like there is only one way to allow users to change their keys: **asset / keystore separation architecture**. Each user has **(i) a "keystore contract"** (on L1 or on one particular L2), which stores the verification key for *all* wallets along with the rules for changing the key, and **(ii) "wallet contracts"** on L1 and many L2s, which read cross-chain to get the verification key.
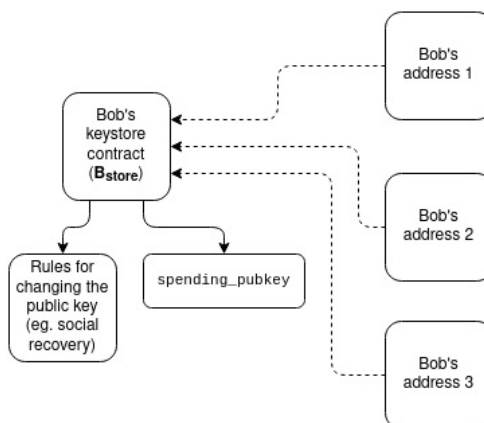
There are two ways to implement this:

- **Light version (check only to update keys)**: each wallet stores the verification key locally, and contains a function which can be called to check a cross-chain proof of the keystore's current state, and update its locally stored verification key to match. When a wallet is used for the first time on a particular L2, calling that function to obtain the current verification key from the keystore is mandatory.
  - **Upside**: uses cross-chain proofs sparingly, so it's okay if cross-chain proofs are expensive. All funds are only spendable with the current keys, so it's still secure.
  - **Downside**: To change the verification key, you have to make an on-chain key change in both the keystore and in every wallet that is already initialized (though not counterfactual ones). This could cost a lot of gas.
- **Heavy version (check for every tx)**: a cross-chain proof showing the key currently in the keystore is necessary for each transaction.
  - **Upside**: less [systemic complexity](#), and keystore updating is cheap.
  - **Downside**: expensive per-tx, so requires much more engineering to make cross-chain proofs acceptably cheap. Also not easily compatible with ERC-4337, which currently does not support cross-contract reading of mutable objects during validation.

What does a cross-chain proof look like?

To show the full complexity, we'll explore the most difficult case: where the keystore is on one L2, and the wallet is on a different L2. If either the keystore on the wallet is on L1, then only half of this design is needed.
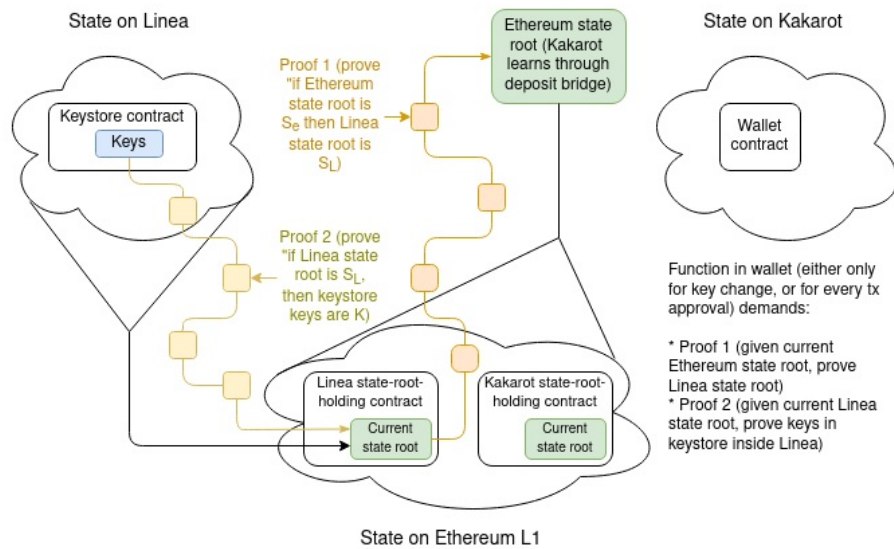
Let's assume that the keystore is on [Linea](#), and the wallet is on [Kakarot](#). A full proof of the keys to the wallet consists of:

- A proof proving the current Linea state root, given the current Ethereum state root that Kakarot knows about
- A proof proving the current keys in the keystore, given the current Linea state root

There are two primary tricky implementation questions here:

1. **What kind of proof do we use**? (Is it Merkle proofs? something else?)
2. **How does the L2 learn the recent L1 (Ethereum) state root** (or, as we shall see, potentially the full L1 state) in the first place? And alternatively, how does the L1 learn the L2 state root?
   - In both cases, **how long are the delays** between something happening on one side, and that thing being provable to the other side?

What kinds of proof schemes can we use?

There are five major options:

- **Merkle proofs**
- **General-purpose ZK-SNARKs**
- **Special-purpose proofs (eg. with KZG)**
- **[Verkle proofs](#)**, which are somewhere between KZG and ZK-SNARKs on both infrastructure workload and cost.
- **No proofs and rely on direct state reading**

In terms of infrastructure work required and cost for users, I rank them roughly as follows:
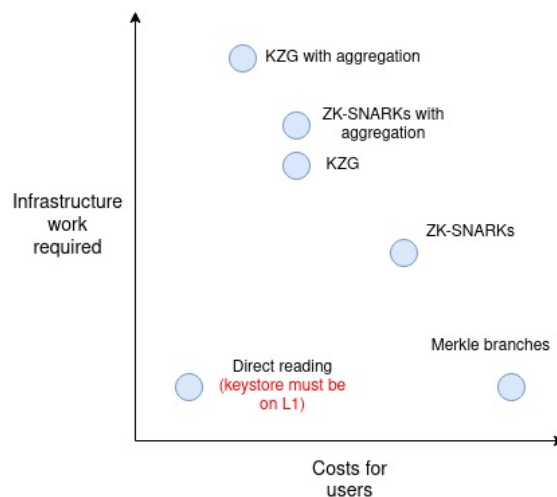
"**Aggregation**" refers to the idea of aggregating all the proofs supplied by users within each block into a big meta-proof that combines all of them. This is possible for SNARKs, and for KZG, but not for Merkle branches (you *can combine Merkle branches a little bit*, but it only saves you `log(txs per block) / log(total number of keystores)`, perhaps 15-30% in practice, so it's probably not worth the cost).

Aggregation only becomes worth it once the scheme has a substantial number of users, so realistically it's okay for a version-1 implementation to leave aggregation out, and implement that for version 2.

How would Merkle proofs work?

This one is simple: follow the diagram in the previous section directly. More precisely, each "proof" (assuming the max-difficulty case of proving one L2 into another L2) would contain:

- **A Merkle branch proving the state-root of the keystore-holding L2, given the most recent state root of Ethereum** that the L2 knows about. The keystore-holding L2's state root is stored at a known storage slot of a known address (the contract on L1 representing the L2), and so the path through the tree could be hardcoded.
- **A Merkle branch proving the current verification keys, given the state-root of the keystore-holding L2**. Here once again, the verification key is stored at a known storage slot of a known address, so the path can be hardcoded.

Unfortunately, Ethereum state proofs are complicated, but there exist libraries for verifying them, and if you use these libraries, this mechanism is not too complicated to implement.

The larger problem is cost. Merkle proofs are long, and Patricia trees are unfortunately ~3.9x longer than necessary (precisely: an ideal Merkle proof into a tree holding `N` objects is `32 * log2(N)` bytes long, and because Ethereum's Patricia trees have 16 leaves per child, proofs for those trees are `32 * 15 * log16(N) ~= 125 * log2(N)` bytes long). In a state with roughly 250 million (~2^{28}) accounts, this makes each proof `125 * 28 = 3500` bytes, or about 56,000 gas, plus extra costs for decoding and verifying hashes.

Two proofs together would end up costing around 100,000 to 150,000 gas (not including signature verification if this is used per-transaction) - significantly more than the current base 21,000 gas per transaction. But **the disparity gets worse if the proof is being verified on L2**. Computation inside an L2 is cheap, because computation is done off-chain and in an ecosystem with much fewer nodes than L1. Data, on the other hand, has to be posted to L1. Hence, the comparison is not 21000 gas vs 150,000 gas; it's 21,000 L2 gas vs 100,000 L1 gas.

We can calculate what this means by looking at comparisons between L1 gas costs and L2 gas costs:

Ethereum Layer–1 is expensive.
How much does it cost to use Layer–2?

How can rollups reduce their fees?
Read our first blog–post "Crunching the Calldata".

Name	Send ETH	Swap tokens
Metis Network ⚠	< $0.01	$0.03 ⌄
Loopring	$0.02	$0.44 ⌄
Polygon zkEVM	$0.03	$0.10 ⌄
Optimism	$0.03	$0.07 ⌄
zkSync Lite	$0.04	$0.09 ⌄
Arbitrum One	$0.05	$0.15 ⌄
Boba Network	$0.07	$0.17 ⌄
StarkNet	$0.12	$0.30 ⌄
Polygon Hermez	$0.25	- ⌄
Ethereum	$0.68	$3.42 ⌄

L1 is currently about 15-25x more expensive than L2 for simple sends, and 20-50x more expensive for token swaps. Simple sends are relatively data-heavy, but swaps are much more computationally heavy. Hence, swaps are a better benchmark to approximate cost of L1 computation vs L2 computation. Taking all this into account, if we assume a 30x cost ratio between L1 computation cost and L2 computation cost, this seems to imply that putting a Merkle proof on L2 will cost the equivalent of perhaps fifty regular transactions.

Of course, using a binary Merkle tree can cut costs by ~4x, but even still, the cost is in most cases going to be too high - and if we're willing to make the sacrifice of no longer being compatible with Ethereum's current hexary state tree, we might as well seek even better options.

How would ZK-SNARK proofs work?

Conceptually, the use of ZK-SNARKs is also easy to understand: you simply replace the Merkle proofs in the diagram above with a ZK-SNARK proving that those Merkle proofs exist. A ZK-SNARK costs ~400,000 gas of computation, and about 400 bytes (compare: 21,000 gas and 100 bytes for a basic transaction, in the future reducible to ~25 bytes with compression). Hence, from a *computational* perspective, a ZK-SNARK costs 19x the cost of a basic transaction today, and from a *data* perspective, a ZK-SNARK costs 4x as much as a basic transaction today, and 16x what a basic transaction may cost in the future.

These numbers are a massive improvement over Merkle proofs, but they are still quite expensive. There are two ways to improve on this: (i) special-purpose KZG proofs, or (ii) aggregation, similar to ERC-4337 aggregation but using more fancy math. We can look into both.

How would special-purpose KZG proofs work?

Warning, this section is much more mathy than other sections. This is because we're going beyond general-purpose tools and building something special-purpose to be cheaper, so we have to go "under the hood" a lot more. If you don't like deep math, skip straight to the next section.

First, a recap of how KZG commitments work:

- We can represent a set of data `[D_1 ... D_n]` with a KZG proof of a *polynomial* derived from the data: specifically, the polynomial P where $P(w) = D\_1, P(w^2) = D\_2 ... P(w^n) = D\_n$. w here is a "root of unity", a value where $w^N = 1$ for some *evaluation domain size* N (this is all done in a [finite field](#)).
- To "commit" to `P`, we create an elliptic curve point $com(P) = P_0 * G + P_1 * S_1 + ... + P_k * S_k$. Here:
  - `G` is the *generator point* of the curve
  - P_i is the i'th-degree coefficient of the polynomial `P`
  - S_i is the i'th point in the *trusted setup*
- To prove `P(z) = a`, we create a *quotient polynomial* $Q = (P - a) / (X - z)$, and create a commitment `com(Q)` to it. It is only possible to create such a polynomial if `P(z)` actually equals a.
- To verify a proof, we check the equation $Q * (X - z) = P - a$ by doing an elliptic curve check on the proof `com(Q)` and the polynomial commitment `com(P)`: we check `e(com(Q), com(X - z)) ?= e(com(P) - com(a), com(1))`

Some key properties that are important to understand are:

- A proof is just the `com(Q)` value, which is 48 bytes
- `com(P₁) + com(P₂) = com(P₁ + P₂)`
- This also means that you can "edit" a value into an existing a commitment. Suppose that we know that `D_i` is currently a, we want to set it to b, and the existing commitment to D is `com(P)`. A commitment to "P, but with $P(w^i) = b$, and no other evaluations changed", then we set `com(new_P) = com(P) + (b-a) * com(L_i)`, where L_i is a the "Lagrange polynomial" that equals 1 at w^i and 0 at other w^j points.
- To perform these updates efficiently, all `N` commitments to Lagrange polynomials (`com(L_i)`) can be pre-calculated and stored by each client. Inside *a contract on-chain* it may be too much to store all `N` commitments, so instead you could make a KZG commitment *to the set of `com(L_i)` (or `hash(com(L_i))` values*, so whenever someone needs to update the tree on-chain they can simply provide the appropriate `com(L_i)` with a proof of its correctness.

Hence, we have a structure where we can just keep adding values to the end of an ever-growing list, though with a certain size limit (realistically, hundreds of millions could be viable). We then use *that* as our data structure to manage (i) a commitment to the list of keys on each L2, stored on that L2 and mirrored to L1, and (ii) a commitment to the list of L2 key-commitments, stored on the Ethereum L1 and mirrored to each L2.

Keeping the commitments updated could either become part of core L2 logic, or it could be implemented without L2 core-protocol changes through deposit and withdraw bridges.
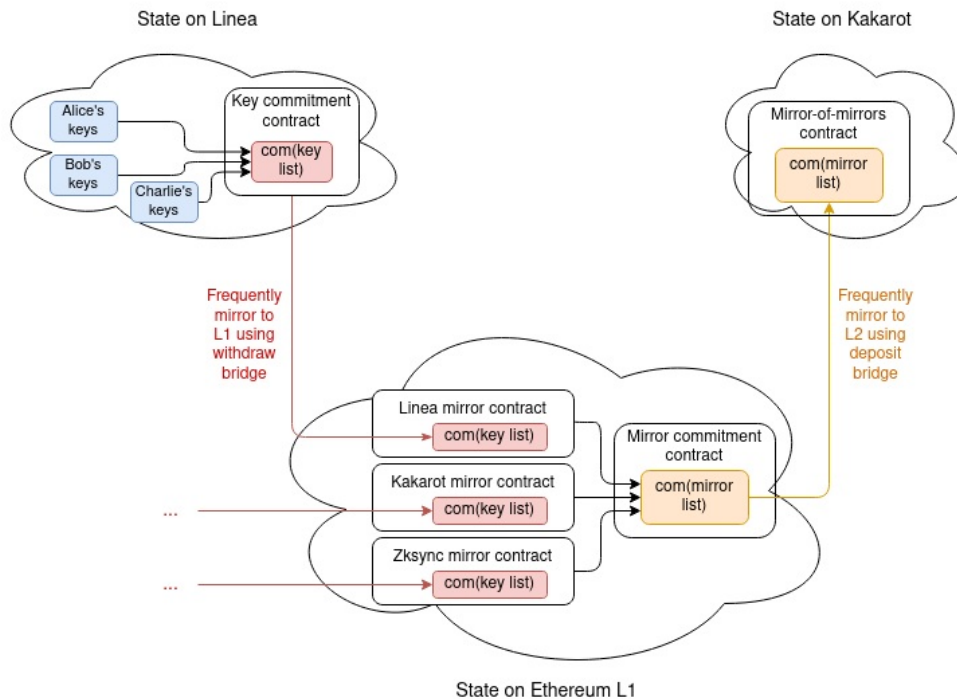
A full proof would thus require:

- The latest `com(key list)` on the keystore-holding L2 (48 bytes)
- KZG proof of `com(key list)` being a value inside `com(mirror_list)`, the commitment to the list of all key list comitments (48 bytes)
- KZG proof of your key in `com(key list)` (48 bytes, plus 4 bytes for the index)

It's actually possible to merge the two KZG proofs into one, so we get a total size of only 100 bytes.

Note one subtlety: because the key list is a list, and not a key/value map like the state is, the key list will have to

assign positions sequentially. The key commitment contract would contain its own internal registry mapping each keystore to an ID, and for each key it would store `hash(key, address of the keystore)` instead of just `key`, to unambiguously communicate to other L2s *which* keystore a particular entry is talking about.

The upside of this technique is that it performs *very well on L2*. The data is 100 bytes, ~4x shorter than a ZK-SNARK and waaaay shorter than a Merkle proof. The computation cost is largely one size-2 pairing check, or [about 119,000 gas](). On L1, data is less important than computation, and so unfortunately KZG is somewhat more expensive than Merkle proofs.

How would Verkle trees work?

Verkle trees essentially involve stacking KZG commitments (or [IPA commitments](), which can be more efficient and use simpler cryptography) on top of each other: to store 2^{48} values, you can make a KZG commitment to a list of 2^{24} values, each of which itself is a KZG commitment to 2^{24} values. Verkle trees are being [strongly considered for the Ethereum state tree](), because Verkle trees can be used to hold key-value maps and not just lists (basically, you can make a size-2^{256} tree but start it empty, only filling in specific parts of the tree once you actually need to fill them).
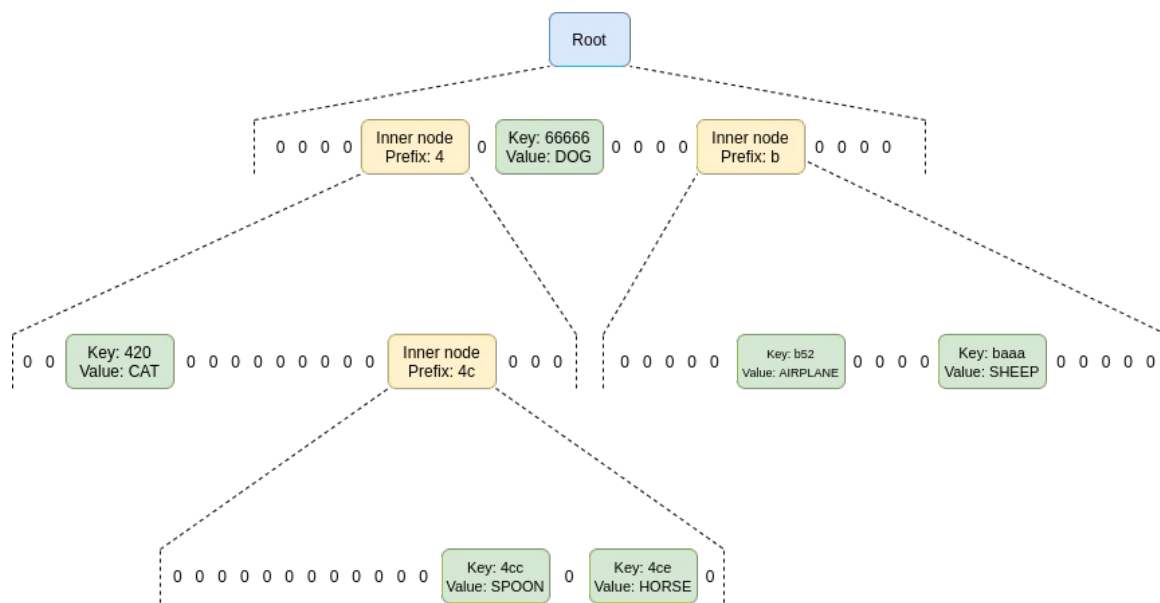
What a Verkle tree looks like. In practice, you might give each node a width of 256 == 2^8 for IPA-based trees, or 2^{24} for KZG-based trees.

Proofs in Verkle trees are somewhat longer than KZG; they might be a few hundred bytes long. They are also difficult to verify, especially if you try to aggregate many proofs into one.

Realistically, Verkle trees should be considered to be like Merkle trees, but more viable without SNARKing (because of the lower data costs), and cheaper with SNARKing (because of lower prover costs).

The largest advantage of Verkle trees is the possibility of harmonizing data structures: Verkle proofs could be used directly over L1 or L2 state, without overlay structures, and using the exact same mechanism for L1 and L2. Once quantum computers become an issue, or once proving Merkle branches becomes efficient enough, Verkle trees could be replaced in-place with a binary hash tree with a suitable SNARK-friendly hash function.

Aggregation

If N users make N transactions (or more realistically, N [ERC-4337]() UserOperations) that need to prove N cross-chain claims, we can save a lot of gas by *aggregating* those proofs: the builder that would be combining those transactions into a block or bundle that goes into a block can create a *single* proof that proves all of those claims simultaneously.

This could mean:

- A ZK-SNARK proof of N Merkle branches
- A [KZG multi-proof]()
- A [Verkle multi-proof]() (or a ZK-SNARK of a multi-proof)

In all three cases, the proofs would only cost a few hundred thousand gas each. The builder would need to make one of these *on each L2* for the users in that L2; hence, for this to be useful to build, the scheme as a whole needs to have enough usage that there are very often at least a few transactions within the same block *on multiple major L2s*.

If ZK-SNARKs are used, the main marginal cost is simply "business logic" of passing numbers around between

contracts, so perhaps a few thousand L2 gas per user. If KZG multi-proofs are used, the prover would need to add 48 gas for each keystore-holding L2 that is used within that block, so the marginal cost of the scheme per user would add another ~800 L1 gas per L2 (not per user) on top. But these costs are much lower than the costs of not aggregating, which inevitably involve over 10,000 L1 gas and hundreds of thousands of L2 gas *per user*. For Verkle trees, you can either use Verkle multi-proofs directly, adding around 100-200 bytes per user, or you can make a ZK-SNARK of a Verkle multi-proof, which has similar costs to ZK-SNARKs of Merkle branches but is significantly cheaper to prove.

From an implementation perspective, it's probably best to have bundlers aggregate cross-chain proofs through the ERC-4337 account abstraction standard. ERC-4337 already has a mechanism for builders to aggregate parts of UserOperations in custom ways. There is even an implementation of this for BLS signature aggregation, which could reduce gas costs on L2 by 1.5x to 3x depending on what other forms of compression are included.
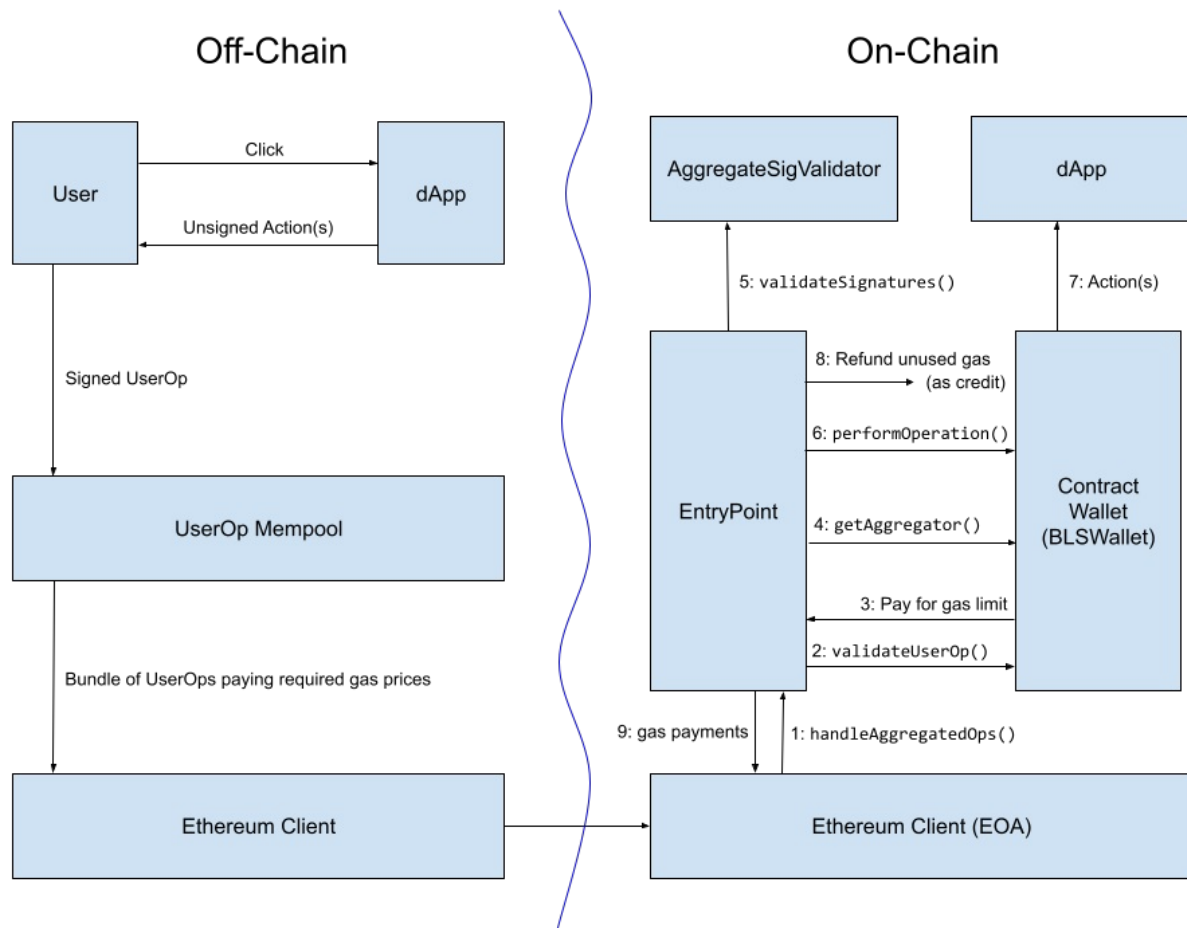
Diagram from a BLS wallet implementation post showing the workflow of BLS aggregate signatures within an earlier version of ERC-4337. The workflow of aggregating cross-chain proofs will likely look very similar.

Direct state reading

A final possibility, and one only usable for L2 reading L1 (and not L1 reading L2), is to **modify L2s to let them make static calls to contracts on L1 directly**.

This could be done with an opcode or a precompile, which allows calls into L1 where you provide the destination address, gas and calldata, and it returns the output, though because these calls are static-calls they cannot actually *change* any L1 state. L2s have to be aware of L1 already to process deposits, so there is nothing fundamental stopping such a thing from being implemented; it is mainly a technical implementation challenge (see: this RFP from Optimism to support static calls into L1).

Notice that if the keystore is on L1, *and* L2s integrate L1 static-call functionality, then no proofs are required at all! However, if L2s don't integrate L1 static-calls, or if the keystore is on L2 (which it may eventually have to be, once L1 gets too expensive for users to use even a little bit), then proofs will be required.

How does L2 learn the recent Ethereum state root?

All of the schemes above require the L2 to access either the recent L1 state root, or the entire recent L1 state. **Fortunately, all L2s have some functionality to access the recent L1 state already**. This is because they need such a functionality to process messages coming in from L1 to the L2, most notably deposits.

And indeed, if an L2 has a deposit feature, then you can use that L2 as-is to move L1 state roots into a contract on the L2: simply have a contract on L1 call the `BLOCKHASH` opcode, and pass it to L2 as a deposit message. The full block header can be received, and its state root extracted, on the L2 side. However, it would be much better for every L2 to have an explicit way to access either the full recent L1 state, or recent L1 state roots, directly.

The main challenge with optimizing how L2s receive recent L1 state roots is simultaneously achieving safety and low latency:

- If L2s implement "direct reading of L1" functionality in a lazy way, **only reading finalized L1 state roots**, then the delay will *normally* **be 15 minutes**, but in the extreme case of inactivity leaks (which you *have to* tolerate), **the delay could be several weeks**.
- L2s absolutely can be designed to read much more recent L1 state roots, but because L1 can revert (even with single slot finality, reverts can happen during inactivity leaks), **L2 would need to be able to revert as well**. This is technically challenging from a software engineering perspective, but at least Optimism already has this capability.
- If you use the **deposit bridge** to bring L1 state roots into L2, then simple **economic viability might require a long time between deposit updates**: if the full cost of a deposit is 100,000 gas, and we assume ETH is at $1800, and fees are at 200 gwei, and L1 roots are brought into L2 *once per day*, that would be a cost of $36 per L2 per day, or $13148 per L2 per year to maintain the system. With a delay of one hour, that goes up to $315,569 per L2 per year. In the best case, a constant trickle of impatient wealthy users covers the updating fees and keep the system up to date for everyone else. In the worst case, some altruistic actor would have to pay for it themselves.
- **"Oracles" (at least, the kind of tech that some defi people *call* "oracles") are *not* an acceptable solution here**: wallet key management is a very security-critical low-level functionality, and so it should depend on at most a few pieces of very simple, cryptographically trustless low-level infrastructure.

Additionally, in the opposite direction (L1s reading L2):

- **On optimistic rollups, state roots take one week to reach L1** because of the fraud proof delay. On ZK rollups it takes a few hours for now because of a combination of proving times and economic limits, though future technology will reduce this.
- **Pre-confirmations (from sequencers, attesters, etc) are not an acceptable solution for L1 reading L2**. Wallet management is a very security-critical low-level functionality, and so the level of security of the L2 -> L1 communication must be absolute: it should not even be possible to push a false L1 state root by taking over the L2 validator set. The only state roots the L1 should trust are state roots that have been accepted as final by the L2's state-root-holding contract on L1.

Some of these speeds for trustless cross-chain operations are unacceptably slow for many defi use cases; for those cases, you do need faster bridges with more imperfect security models. For the use case of updating wallet keys, however, longer delays are more acceptable: you're not *delaying transactions* by hours, you're delaying *key changes*. You'll just have to keep the old keys around longer. If you're changing keys because keys are stolen, then you do have a significant period of vulnerability, but this can be mitigated, eg. by wallets having a `freeze` function.

Ultimately, the best latency-minimizing solution is for L2s to implement direct reading of L1 state roots in an optimal way, where each L2 block (or the state root computation log) contains a pointer to the most recent L1 block, so if L1 reverts, L2 can revert as well. Keystore contracts should be placed either on mainnet, or on L2s that are ZK-rollups and so can quickly commit to L1.
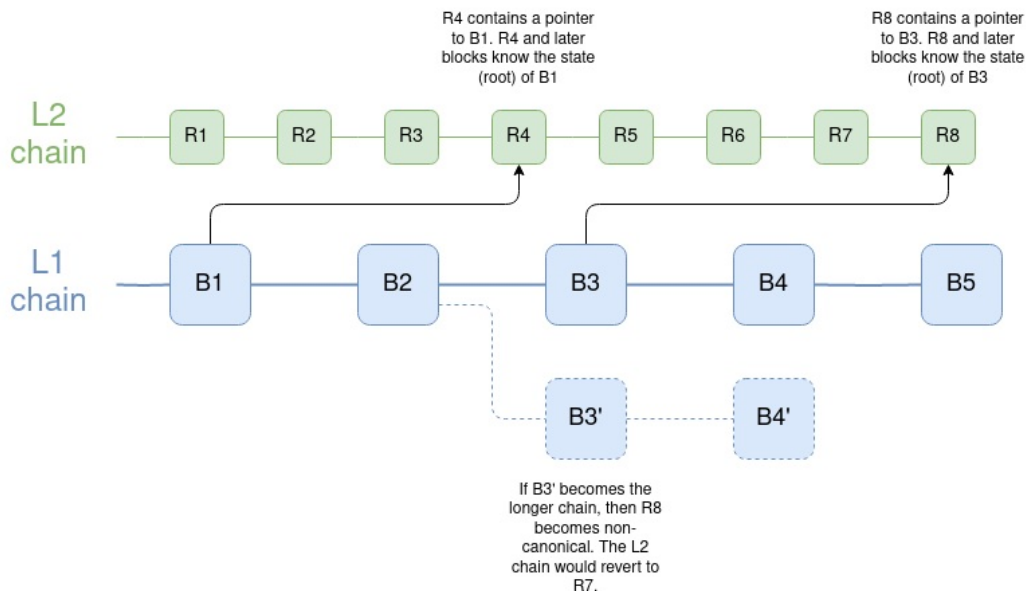
Blocks of the L2 chain can have dependencies on not just previous L2 blocks, but also on an L1 block. If L1 reverts past such a link, the L2 reverts too. It's worth noting that this is also how an earlier (pre-Dank) version of sharding was envisioned to work; see here for code.

How much connection to Ethereum does another chain need to hold wallets whose keystores are rooted on Ethereum or an L2?

Surprisingly, not that much. It actually does not even need to be a rollup: if it's an L3, or a validium, then it's okay to hold wallets there, as long as you hold keystores either on L1 or on a ZK rollup. The thing that you *do* need is for the chain to have direct access to Ethereum state roots, and **a technical and social commitment to be willing to reorg if Ethereum reorgs, and hard fork if Ethereum hard forks**.

One interesting research problem is identifying to what extent it is possible for a chain to have this form of connection to *multiple* other chains (eg. Ethereum and Zcash). Doing it naively is possible: your chain could agree to reorg if Ethereum *or* Zcash reorg (and hard fork if Ethereum *or* Zcash hard fork), but then your node operators and your community more generally have double the technical and political dependencies. Hence such a technique could be used to connect to a few other chains, but at increasing cost. Schemes based on ZK bridges have attractive technical properties, but they have the key weakness that they are not robust to 51% attacks or hard forks. There may be more clever solutions.

Preserving privacy

Ideally, we also want to preserve privacy. If you have many wallets that are managed by the same keystore, then we want to make sure:

- It's not publicly known that those wallets are all connected to each other.
- Social recovery guardians don't learn what the addresses are that they are guarding.

This creates a few issues:

- We cannot use Merkle proofs directly, because they do not preserve privacy.
- If we use KZG or SNARKs, then the proof needs to provide a blinded version of the verification key, without revealing the location of the verification key.
- If we use aggregation, then the aggregator should not learn the location in plaintext; rather, the aggregator should receive blinded proofs, and have a way to aggregate those.
- We can't use the "light version" (use cross-chain proofs only to update keys), because it creates a privacy leak: if many wallets get updated at the same time due to an update procedure, the timing leaks the information that those wallets are likely related. So we have to use the "heavy version" (cross-chain proofs for each transaction).

With SNARKs, the solutions are conceptually easy: proofs are information-hiding by default, and the aggregator needs to produce a recursive SNARK to prove the SNARKs.
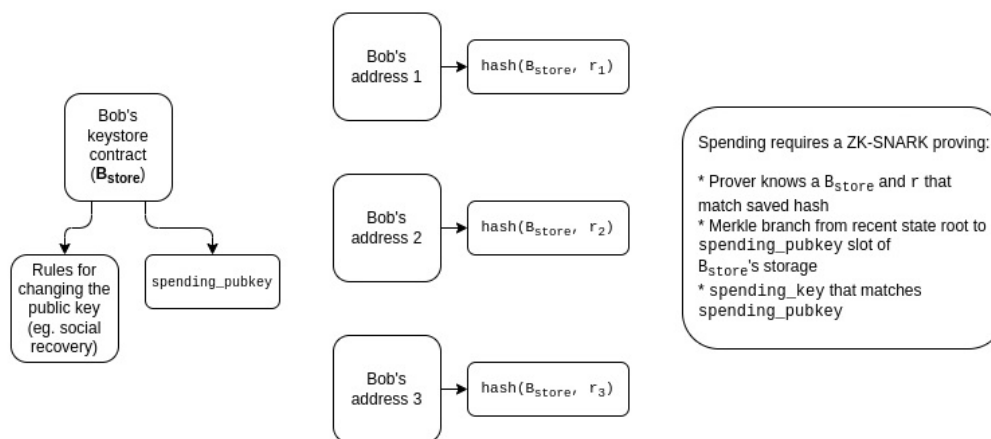
The main challenge of this approach today is that aggregation requires the aggregator to create a recursive SNARK, which is currently quite slow.

With KZG, we can use [this work on non-index-revealing KZG proofs](#) (see also: a more formalized version of that work in [the Caulk paper](#)) as a starting point. Aggregation of blinded proofs, however, is an open problem that requires more attention.

Directly reading L1 from inside L2, unfortunately, does not preserve privacy, though implementing direct-reading functionality is still very useful, both to minimize latency and because of its utility for other applications.

Summary

- To have cross-chain social recovery wallets, the most realistic workflow is a wallet that maintains a **keystore** in one location, and **wallets** in many locations, where wallet reads the keystore either (i) to update their local view of the verification key, or (ii) during the process of verifying each transaction.
- A key ingredient of making this possible is **cross-chain proofs**. We need to optimize these proofs hard. Either **ZK-SNARKs**, waiting for **Verkle proofs**, or a **custom-built KZG solution**, seem like the best options.
- In the longer term, **aggregation protocols** where bundlers generate aggregate proofs as part of creating a bundle of all the UserOperations that have been submitted by users will be necessary to minimize costs. This should probably be integrated into the **ERC-4337** ecosystem, though changes to ERC-4337 will likely be required.
- L2s should be optimized to **minimize the latency of reading L1 state** (or at least the state root) from inside the L2. L2s **directly reading L1 state** is ideal and can save on proof space.
- Wallets can be not just on L2s; **you can also put wallets on systems with lower levels of connection to Ethereum** (L3s, or even separate chains that only agree to include Ethereum state roots and reorg or hard fork when Ethereum reorgs or hardforks).
- However, **keystores should be either on L1 or on high-security ZK-rollup L2** . Being on L1 saves a lot of complexity, though in the long run even that may be too expensive, hence the need for keystores on L2.
- Preserving **privacy** will require additional work and make some options more difficult. However, we should probably move toward privacy-preserving solutions anyway, and at the least make sure that anything we propose is forward-compatible with preserving privacy.

The Three Transitions

2023 Jun 09

Special thanks to Dan Finlay, Karl Floersch, David Hoffman, and the Scroll and SoulWallet teams for feedback and review and suggestions.

As Ethereum transitions from a young experimental technology into a mature tech stack that is capable of actually bringing an open, global and permissionless experience to average users, there are three major technical transitions that the stack needs to undergo, roughly simultaneously:

- **The L2 scaling transition** - everyone [moving to rollups](#)
- **The wallet security transition** - everyone moving to [smart contract wallets](#)
- **The privacy transition** - making sure privacy-preserving funds transfers are available, and making sure all of the *other* gadgets that are being developed (social recovery, identity, reputation) are privacy-preserving
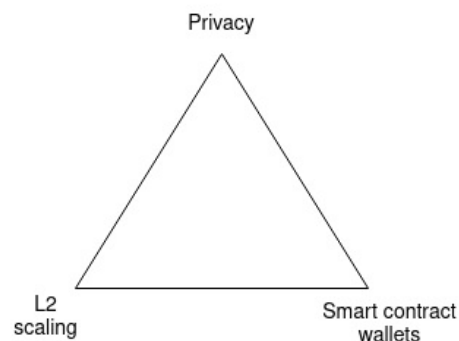
The ecosystem transition triangle. **You can only pick 3 out of 3.**

Without the first, Ethereum fails because each transaction costs $3.75 ($82.48 if we have another bull run), and every product aiming for the mass market inevitably forgets about the chain and adopts centralized workarounds for everything.

Without the second, Ethereum fails because users are uncomfortable storing their funds (and non-financial assets), and everyone moves onto centralized exchanges.

Without the third, Ethereum fails because having all transactions (and POAPs, etc) available publicly for literally anyone to see is far too high a privacy sacrifice for many users, and everyone moves onto centralized solutions that at least somewhat hide your data.

These three transitions are crucial for the reasons above. But they are also challenging because of the intense coordination involved to properly resolve them. It's not just features of the protocol that need to improve; in some cases, the way that we interact with Ethereum needs to change pretty fundamentally, requiring deep changes from applications and wallets.

The three transitions will radically reshape the relationship between *users* and *addresses*

In an L2 scaling world, users are going to exist on lots of L2s. Are you a member of ExampleDAO, which lives on Optimism? Then you have an account on Optimism! Are you holding a CDP in a stablecoin system on ZkSync? Then you have an account on ZkSync! Did you *once* go try some application that happened to live on Kakarot? Then you have an account on Kakarot! The days of a user having only one address will be gone.

I have ETH in four places, according to my Brave Wallet view. And yes, Arbitrum and Arbitrum Nova are different. Don't worry, it will get more confusing over time!

Smart contract wallets add more complexity, by making it much more difficult to have *the same* address across L1 and the various L2s. Today, most users are using *externally owned accounts*, whose address is literally a hash of the public key that is used to verify signatures - so nothing changes between L1 and L2. With smart contract wallets, however, keeping one address becomes more difficult. Although a lot of work has been done to *try* to make addresses be hashes of code that can be equivalent across networks, most notably [CREATE2](#) and the [ERC-2470 singleton factory](#), it's difficult to make this work perfectly. Some L2s (eg. "[type 4](#) ZK-EVMs") are not *quite* EVM equivalent, often using Solidity or an intermediate assembly instead, preventing hash equivalence. And even when you can have hash equivalence, the possibility of wallets changing ownership through key changes creates [other unintuitive consequences](#).

Privacy requires each user to have *even more* addresses, and may even change *what kinds* of addresses we're dealing with. If [stealth address](#) proposals become widely used, instead of each user having only a few addresses, or one address per L2, users might have *one address per transaction*. Other privacy schemes, even existing ones such as Tornado Cash, change how assets are stored in a different way: *many users' funds are stored in the same smart contract* (and hence at the same address). To send funds to a specific user, users will need to rely on the privacy scheme's own internal addressing system.

As we've seen, **each of the three transitions weaken the "one user ~= one address" mental model in different ways**, and some of these effects feed back into the complexity of executing the transitions. Two particular points of complexity are:

1. **If you want to pay someone, how will you get the information on *how* to pay them?**
2. **If users have many assets stored in different places across different chains, how do they do key changes and [social recovery](#)?**

The three transitions and on-chain payments (and identity)

I have coins on Scroll, and I want to pay for coffee (if the "I" is literally me, the writer of this article, then "coffee" is of course a metonymy for "green tea"). You are selling me the coffee, but you are only set up to receive coins on Taiko. Wat do?

There are basically two solutions:

1. Receiving wallets (which could be merchants, but also could just be regular individuals) try really hard to support *every* L2, and have some automated functionality for consolidating funds asynchronously.
2. The recipient provides their L2 alongside their address, and the sender's wallet automatically routes funds to the destination L2 through some cross-L2 bridging system.

Of course, these solutions can be combined: the recipient provides the *list* of L2s they're willing to accept, and the sender's wallet figures out payment, which could involve either a direct send if they're lucky, or otherwise a cross-L2 bridging path.

But this is only one example of a key challenge that the three transitions introduce: **simple actions like *paying someone* start to require a lot more information than just a 20-byte address**.

A transition to smart contract wallets is fortunately not a large burden on the addressing system, but there are still some technical issues in other parts of the application stack that need to be worked through. Wallets will need to be updated to make sure that they do not send only 21000 gas along with a transaction, and it will be even more important to ensure that the payment *receiving* side of a wallet tracks not only ETH transfers from EOAs, but also ETH sent by smart contract code. Apps that rely on the assumption that address ownership is *immutable* (eg. NFTs that ban smart contracts to enforce royalties) will have to find other ways of achieving their goals. Smart contract wallets will also make some things *easier* - notably, if someone receives *only* a non-ETH ERC20 token, they will be able to use ERC-4337 paymasters to pay for gas with that token.

Privacy, on the other hand, once again poses major challenges that we have not really dealt with yet. The original Tornado Cash did not introduce any of these issues, because it did not support internal transfers: users could only deposit into the system and withdraw out of it. Once you *can* make internal transfers, however, users will need to use the internal addressing scheme of the privacy system. In practice, a user's "payment information" would need to contain both (i) some kind of "spending pubkey", a commitment to a secret that the recipient could use to spend, and (ii) some way for the sender to send encrypted information that only the recipient can decrypt, to help the recipient discover the payment.

Stealth address protocols rely on a concept of **meta-addresses**, which work in this way: one part of the meta-address is a blinded version of the sender's spending key, and another part is the sender's encryption key (though a minimal implementation could set those two keys to be the same).
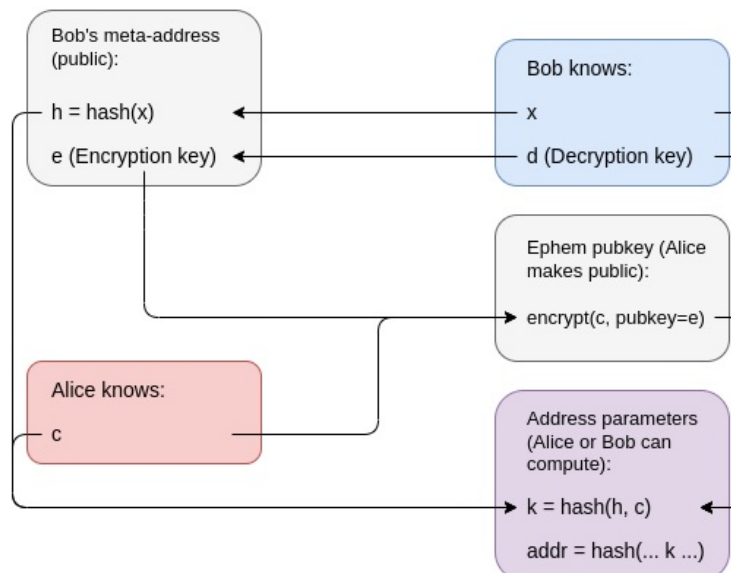
Schematic overview of an abstract stealth address scheme based on encryption and ZK-SNARKs.

A key lesson here is that **in a privacy-friendly ecosystem, a user will have both spending pubkeys and encryption pubkeys, and a user's "payment information" will have to include both keys**. There are also good reasons other than payments to expand in this direction. For example, if we want Ethereum-based encrypted email, users will need to publicly provide some kind of encryption key. In "EOA world", we could re-use account keys for this, but in a safe smart-contract-wallet world, we probably should have more explicit functionality for this. This would also help in making Ethereum-based identity more compatible with non-Ethereum decentralized privacy ecosystems, most notably PGP keys.

The three transitions and key recovery

The default way to implement key changes and social recovery in a many-address-per-user world is to simply have users run the recovery procedure on each address separately. This can be done in one click: the wallet can include software to execute the recovery procedure across all of a user's addresses at the same time. However, even with such UX simplifications, naive multi-address recovery has three issues:

1. **Gas cost impracticality**: this one is self-explanatory.
2. **Counterfactual addresses**: addresses for which the smart contract has not yet been published (in

practice, this will mean an account that you have not yet sent funds from). You as a user have a potentially unlimited number of counterfactual addresses: one or more on every L2, including L2s that do not yet exist, and a whole other infinite set of counterfactual addresses arising from stealth address schemes.

3. **Privacy**: if a user intentionally has many addresses to avoid linking them to each other, they certainly do not want to publicly link all of them by recovering them at or around the same time!

Solving these problems is hard. Fortunately, there is a somewhat elegant solution that performs reasonably well: **an architecture that separates verification logic and asset holdings**.
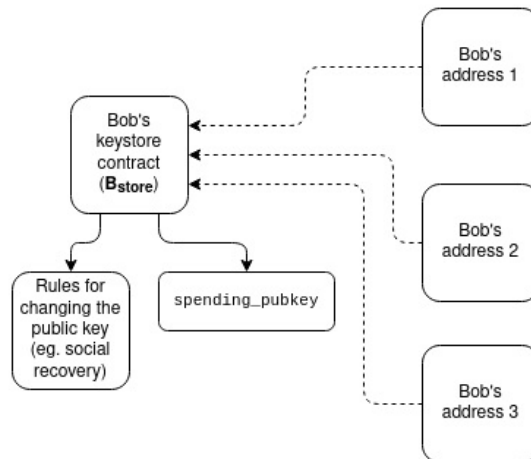
Each user has a **keystore contract**, which exists in *one location* (could either be mainnet or a specific L2). Users then have addresses on different L2s, where the verification logic of each of those addresses is *a pointer to the keystore contract*. Spending from those addresses would require a proof going into the keystore contract showing the *current* (or, more realistically, *very recent*) spending public key.

The proof could be implemented in a few ways:

- **Direct read-only L1 access inside the L2**. It's possible to modify L2s to give them a way to directly read L1 state. If the keystore contract is on L1, this would mean that contracts inside L2 can access the keystore "for free"
- **Merkle branches**. Merkle branches can prove L1 state to an L2, or L2 state to an L1, or you can combine the two to prove parts of the state of one L2 to another L2. The main weakness of Merkle proofs is high gas costs due to proof length: potentially 5 kB for a proof, though this will reduce to < 1 kB in the future due to [Verkle trees](#).
- **ZK-SNARKs**. You can reduce data costs by using a ZK-SNARK of a Merkle branch instead of the branch itself. It's possible to build off-chain aggregation techniques (eg. on top of [EIP-4337](#)) to have one single ZK-SNARK verify all cross-chain state proofs in a block.
- **KZG commitments**. Either L2s, or schemes built on top of them, could introduce a sequential addressing system, allowing proofs of state inside this system to be a mere 48 bytes long. Like with ZK-SNARKs, a [multiproof scheme](#) could merge all of these proofs into a single proof per block.
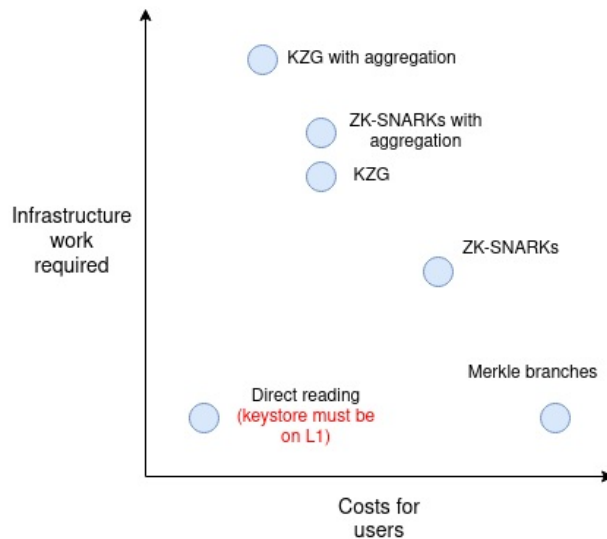
If we want to avoid making one proof per transaction, we can implement a lighter scheme that only requires a cross-L2 proof for recovery. *Spending* from an account would depend on a spending key whose corresponding pubkey is stored within that account, but *recovery* would require a transaction that copies over the current `spending_pubkey` in the keystore. Funds in counterfactual addresses are safe even if your old keys are not: "activating" a counterfactual address to turn it into a working contract would require making a cross-L2 proof to copy over the current `spending_pubkey`. [This thread on the Safe forums](#) describes how a similar architecture might work.

To add *privacy* to such a scheme, then we just encrypt the pointer, and we do all of our proving inside ZK-SNARKs:
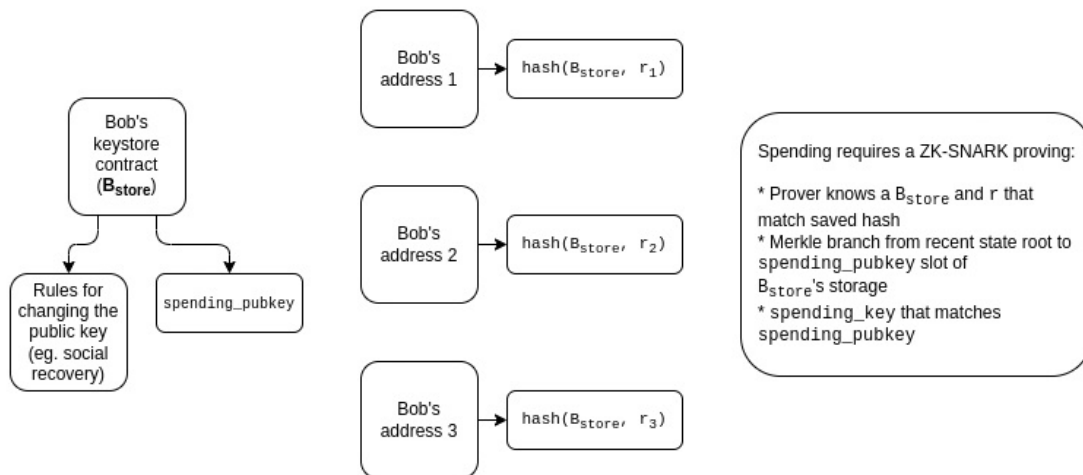
With more work (eg. using [this work](#) as a starting point), we could also strip out most of the complexity of ZK-SNARKs and make a more bare-bones KZG-based scheme.

These schemes can get complex. On the plus side, there are many potential synergies between them. For example, the concept of "keystore contracts" could also be a solution to the challenge of "addresses" mentioned in the previous section: if we want users to have persistent addresses, that do not change every time the user updates a key, we could put stealth meta-addresses, encryption keys, and other information into the keystore contract, and use the address of the keystore contract as a user's "address".

Lots of secondary infrastructure needs to update

Using ENS is expensive. Today, in June 2023, the situation is not too bad: the transaction fee is significant, but it's still comparable to the ENS domain fee. [Registering zuzalu.eth](#) cost me roughly $27, of which $11 was transaction fees. But if we have another bull market, fees will skyrocket. Even without ETH price increases, gas fees returning to 200 gwei would raise the tx fee of a domain registration to $104. And so if we want people to actually use ENS, especially for use cases like decentralized social media where users

demand nearly-free registration (and the ENS domain fee is not an issue because these platforms offer their users sub-domains), we need ENS to work on L2.

Fortunately, the ENS team has stepped up, and ENS on L2 is actually happening! ERC-3668 (aka "the CCIP standard"), together with ENSIP-10, provide a way to have ENS subdomains on *any* L2 automatically be verifiable. The CCIP standard requires setting up a smart contract that describes a method for verifying *proofs* of data on L2, and a domain (eg. Optinames uses ecc.eth) can be put under the control of such a contract. Once the CCIP contract controls ecc.eth on L1, accessing some subdomain.ecc.eth will automatically involve finding and verifying a proof (eg. Merkle branch) of the state in L2 that actually stores that particular subdomain.
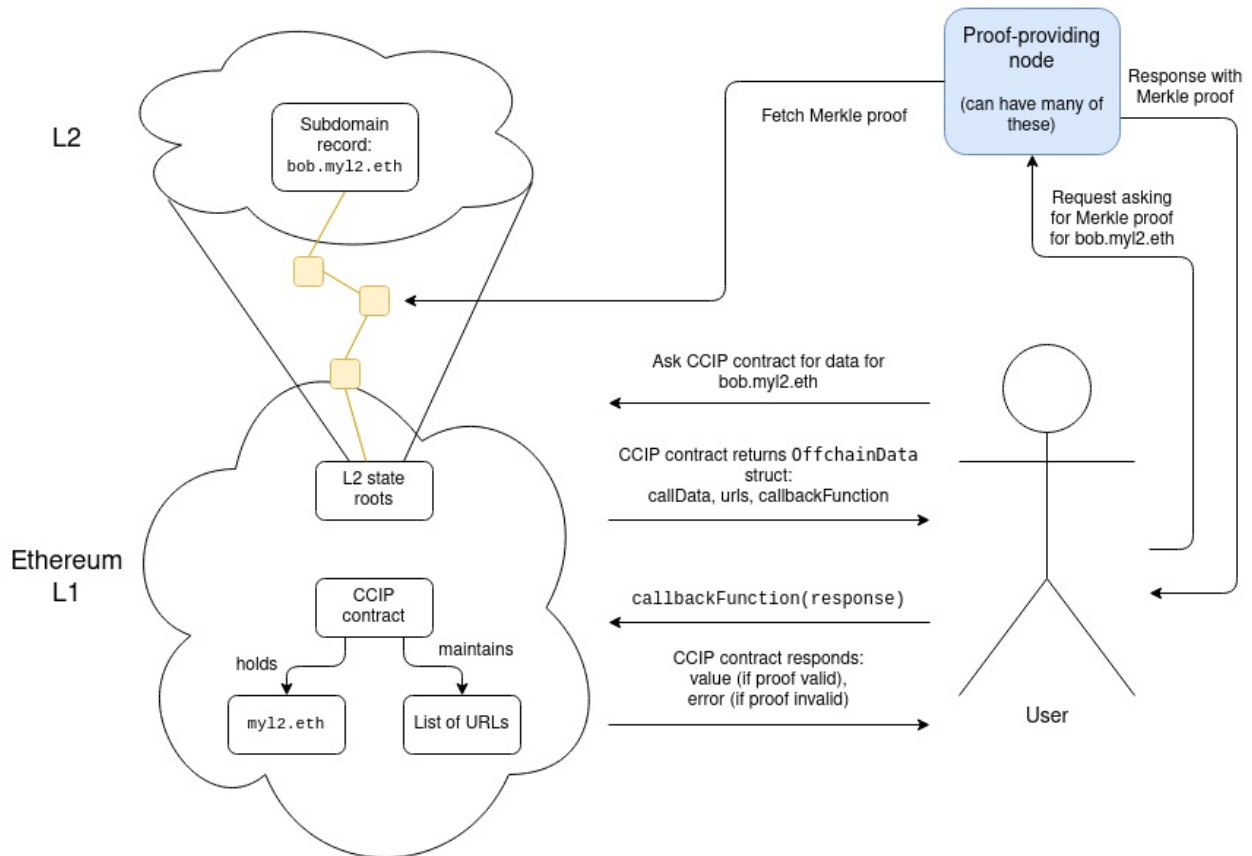
Actually fetching the proofs involves going to a list of URLs stored in the contract, which admittedly *feels* like centralization, though I would argue it really isn't: it's a 1-of-N trust model (invalid proofs get caught by the verification logic in the CCIP contract's callback function, and as long as even *one* of the URLs returns a valid proof, you're good). The list of URLs could contain dozens of them.

The ENS CCIP effort is a success story, and it should be viewed as a sign that radical reforms of the kind that we need are actually possible. But there's a lot more application-layer reform that will need to be done. A few examples:

- Lots of dapps depend on users providing **off-chain *signatures***. With externally-owned accounts (EOAs), this is easy. ERC-1271 provides a standardized way to do this for smart contract wallets. However, lots of dapps still don't support ERC-1271; they will need to.
- **Dapps that use "is this an EOA?" to discriminate between users and contracts (eg. to prevent transfer or enforce royalties) will break**. In general, I advise against attempting to find a purely technical solution here; figuring out whether or not a particular transfer of cryptographic control is a transfer of beneficial ownership is a difficult problem and probably not solvable without resolving to some off-chain community-driven mechanisms. Most likely, applications will have to rely less on preventing transfers and more on techniques like Harberger taxes.
- **How wallets interact with spending and encryption keys will have to be improved.** Currently, wallets often use deterministic signatures to generate application-specific keys: signing a standard nonce (eg. the hash of the application's name) with an EOA's private key generates a deterministic value that cannot be generated without the private key, and so it's secure in a purely technical sense.

However, these techniques are "opaque" to the wallet, preventing the wallet from implementing user-interface level security checks. In a more mature ecosystem, signing, encryption and related functionalities will have to be handled by wallets more explicitly.

- **Light clients (eg. Helios) will have to verify L2s and not just the L1**. Today, light clients focus on checking the validity of the L1 headers (using the light client sync protocol), and verifying Merkle branches of L1 state and transactions rooted in the L1 header. Tomorrow, they will *also* need to verify a proof of L2 state rooted in the state root stored in the L1 (a more advanced version of this would actually look at *L2 pre-confirmations*).

Wallets will need to secure both *assets* and *data*

Today, wallets are in the business of securing *assets*. Everything lives on-chain, and the only thing that the wallet needs to protect is the private key that is *currently* guarding those assets. If you change the key, you can safely publish your previous private key on the internet the next day. In a ZK world, however, this is no longer true: the wallet is not just protecting authentication credentials, it's also holding your *data*.

We saw the first signs of such a world with Zupass, the ZK-SNARK-based identity system that was used at Zuzalu. Users had a private key that they used to authenticate to the system, which could be used to make basic proofs like "prove I'm a Zuzalu resident, without revealing which one". But the Zupass system also began to have other apps built on top, most notably *stamps* (Zupass's version of POAPs).

One of my many Zupass stamps, confirming that I am a proud member of Team Cat.

The key feature that stamps offer over POAPs is that stamps are private: you hold the data locally, and you only ZK-prove a stamp (or some computation over the stamps) to someone if you want them to have that information about you. But this creates added risk: if you lose that information, you lose your stamps.

Of course, the problem of holding data can be reduced to the problem of holding a single encryption key: some third party (or even the chain) can hold an encrypted copy of the data. This has the convenient advantage that actions you take don't change the encryption key, and so do not require any interactions with the system holding your encryption key safe. But even still, **if you lose your encryption key, you lose everything**. And on the flip side, **if someone *sees* your encryption key, they see everything that was encrypted to that key**.

Zupass's de-facto solution was to encourage people to store their key on multiple devices (eg. laptop and phone), as the chance that they would lose access to all devices at the same time is tiny. We could go further, and use secret sharing to store the key, split between multiple guardians.

This kind of social recovery via MPC is not a sufficient solution for *wallets*, because it means that not only current guardians but also previous guardians could collude to steal your assets, which is an unacceptably high risk. But privacy leaks are generally a lower risk than total asset loss, and someone with a high-privacy-demanding use case could always accept a higher risk of loss by not backing up the key associated with those privacy-demanding actions.

To avoid overwheming the user with a byzantine system of multiple recovery paths, wallets that support social recovery will likely need to manage both recovery of assets *and* recovery of encryption keys.

Back to identity

One of the common threads of these changes is that the concept of an "address", a cryptographic identifier that you use to represent "you" on-chain, will have to radically change. **"Instructions for how to interact with me" would no longer just be an ETH address; they would have to be, in some form, some combination of multiple addresses on multiple L2s, stealth meta-addresses, encryption keys, and other data**.

One way to do this is to make ENS your identity: your ENS record could just contain all of this information, and if you send someone bob.eth (or bob.ecc.eth, or...), they could look up and see everything about how to pay and interact with you, including in the more complicated cross-domain and privacy-preserving ways.

But this ENS-centric approach has two weaknesses:

- **It ties too many things to *your name***. Your name is not you, your name is one of many attributes of you. It should be possible to change your name without moving over your entire identity profile and updating a whole bunch of records across many applications.
- **You can't have trustless counterfactual names**. One key UX feature of any blockchain is the ability to send coins to people who have not interacted with the chain yet. Without such a functionality, there is a catch-22: interacting with the chain requires paying transaction fees, which requires... already having coins. ETH addresses, including smart contract addresses with CREATE2, have this feature. ENS names don't, because if two Bobs both decide off-chain that they are bob.ecc.eth, there's no way to choose which one of them gets the name.

One possible solution is to put more things into the **keystore contract** mentioned in the architecture earlier in this post. The keystore contract could contain all of the various information about you and how to interact with you (and with CCIP, some of that info could be off-chain), and users would use their keystore contract as their primary identifier. But the actual *assets* that they receive would be stored in all kinds of different places. Keystore contracts are not tied to a name, and they are counterfactual-friendly: you can generate an address that can provably only be initialized by a keystore contract that has certain fixed initial parameters.

Another category of solutions has to do with abandoning the concept of user-facing addresses altogether, in a similar spirit to the Bitcoin payment protocol. One idea is to rely more heavily on direct communication channels between the sender and the recipient; for example, the sender could send a claim link (either as an explicit URL or a QR code) which the recipient could use to accept the payment however they wish.
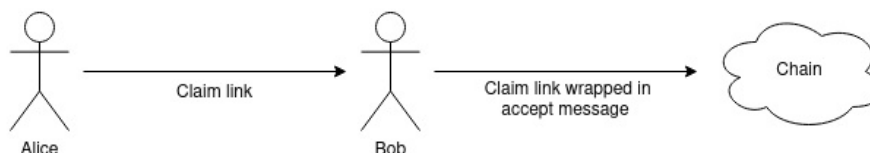
Regardless of whether the sender or the recipient acts first, greater reliance on wallets directly generating up-to-date payment information in real time could reduce friction. That said, persistent identifiers are convenient (especially with ENS), and the assumption of direct communication between sender and recipient is a really tricky one in practice, and so we may end up seeing a combination of different techniques.

In all of these designs, keeping things both decentralized and understandable to users is paramount. We need to make sure that users have easy access to an up-to-date view of what their current assets are and what messages have been published that are intended for them. These views should depend on open tools,

not proprietary solutions. It will take hard work to avoid the greater complexity of payment infrastructure from turning into an opaque "tower of abstraction" where developers have a hard time making sense of what's going on and adapting it to new contexts. Despite the challenges, achieving scalability, wallet security, and privacy for regular users is crucial for Ethereum's future. It is not just about technical feasibility but about actual accessibility for regular users. We need to rise to meet this challenge.

Don't overload Ethereum's consensus

2023 May 21

Special thanks to Karl Floersch and Justin Drake for feedback and review

The Ethereum network's consensus is one of the most highly secured cryptoeconomic systems out there. 18 million ETH (~$34 billion) worth of validators finalize a block every 6.4 minutes, running many different implementations of the protocol for redundancy. And if the *cryptoeconomic* consensus fails, whether due to a bug or an intentional 51% attack, a vast community of many thousands of developers and many more users are watching carefully to make sure the chain recovers correctly. Once the chain recovers, protocol rules ensure that attackers will likely be heavily penalized.

Over the years there have been a number of ideas, usually at the thought experiment stage, to also use the Ethereum validator set, and perhaps even the Ethereum social consensus, *for other purposes*:

- **The ultimate oracle**: a proposal where users can vote on what facts are true by sending ETH, with a SchellingCoin mechanism: everyone who sent ETH to vote for the majority answer gets a proportional share of all the ETH sent to vote for the minority answer. The description continues: "So in principle this is an symmetric game. What breaks the symmetry is that a) the truth is the natural point to coordinate on and more importantly b) the people betting on the truth can make a credible thread of forking Ethereum if they loose."
- **Re-staking**: a set of techniques, used by many protocols including EigenLayer, where Ethereum stakers can simultaneously use their stake as a deposit in another protocol. In some cases, if they misbehave according to the other protocol's rules, their deposit also gets slashed. In other cases, there are no in-protocol incentives and stake is simply used to vote.
- **L1-driven recovery of L2 projects**: it has been proposed on many occasions that if an L2 has a bug, the L1 could fork to recover it. One recent example is this design for using L1 soft forks to recover L2 failures.

The purpose of this post will be to explain in detail the argument why, in my view, *a certain subset* of these techniques brings high systemic risks to the ecosystem and should be discouraged and resisted.

These proposals are generally made in a well-intentioned way, and so the goal is not to focus on individuals or projects; rather, the goal is to focus on techniques. The general rule of thumb that this post will attempt to defend is as follows: **dual-use of validator staked ETH, while it has some risks, is fundamentally fine, but attempting to "recruit" Ethereum social consensus for your application's own purposes is not.**

Examples of the distinction between re-using validators (low-risk) and overloading social consensus (high-risk)

- Alice creates a web3 social network where if you cryptographically prove that you control the key of an active Ethereum validator, you automatically gain "verified" status. **Low-risk**.
- Bob cryptographically proves that he controls the key of ten active Ethereum validators as a way of proving that he has enough wealth to satisfy some legal requirement. **Low-risk**.
- Charlie claims to have disproven the twin primes conjecture, and claims to know the largest `p` such that `p` and `p+2` are both prime. He changes his staking withdrawal address to a smart contract where anyone can submit a claimed counterexample `q > p`, along with a SNARK proving that `q` and `q+2` are both prime. If someone makes a valid claim, then Charlie's validator is forcibly exited, and the submitter gets whatever of Charlie's ETH is left. **Low-risk**.
- Dogecoin decides to switch to proof of stake, and to increase the size of its security pool it allows Ethereum stakers to "dual-stake" and simultaneously join its validator set. To do so, Ethereum stakers would have to change their staking withdrawal address to a smart contract where anyone can submit a proof that they violated the *Dogecoin staking rules*. If someone does submit such a proof, then the staker's validator is forcibly exited, and whatever of their ETH is left is used to buy-and-burn DOGE. **Low-risk**.
- eCash does the same as Dogecoin, but the project leaders further announce: if the majority of participating ETH validators collude to censor *eCash transactions*, they expect that the Ethereum community will hard-fork to delete those validators. They argue that it will be in

Ethereum's interest to do so as those validators are proven to be malicious and unreliable. **High-risk**.

- Fred creates an ETH/USD price oracle, which functions by allowing Ethereum validators to participate and vote. There are no incentives. **Low-risk**.
- George creates an ETH/USD price oracle, which functions by allowing ETH holders to participate and vote. To protect against laziness and creeping bribes, they add an incentive mechanism where the participants that give an answer within 1% of the median answer get 1% of the ETH of any participants that gave an answer further than 1% from the median. When asked "what if someone credibly offers to bribe all the participants, everyone starts submitting the wrong answer, and so honest people get 10 million of their ETH taken away?", George replies: then Ethereum will have to fork out the bad participants' money. **High-risk**.
  - George conspicuously stays away from making replies. **Medium-high risk** (as the project could create incentives to attempt such a fork, and hence the expectation that it will be attmpted, even without formal encouragement)
  - George replies: "then the attacker wins, and we'll give up on using this oracle". **Medium-low risk** (not quite "low" only because the mechanism does create a large set of actors who in a 51% attack might be incentivized to indepently advocate for a fork to protect their deposits)
- Hermione creates a successful layer 2, and argues that because her layer 2 is the largest, it is inherently the most secure, because if there is a bug that causes funds to be stolen, the losses will be so large that the community will have no choice but to fork to recover the users' funds. **High-risk**.

If you're designing a protocol where, even if everything completely breaks, the losses are kept contained to the validators and users who opted in to participating in and using your protocol, this is low-risk. If, on the other hand, you have the intent to rope in the broader Ethereum ecosystem social consensus to fork or reorg to solve your problems, this is high-risk, and I argue that we should strongly resist all attempts to create such expectations.

A middle ground is situations that start off in the low-risk category but give their participants incentives to slide into the higher-risk category; SchellingCoin-style techniques, especially mechanisms with heavy penalties for deviating from the majority, are a major example.

So what's so wrong with stretching Ethereum consensus, anyway?

It is the year 2025. Frustrated with the existing options, a group has decided to make a new ETH/USD price oracle, which works by allowing validators to vote on the price every hour. If a validator votes, they would be unconditionally rewarded with a portion of fees from the system. But soon participants became lazy: they connected to centralized APIs, and when those APIs got cyber-attacked, they either dropped out or started reporting false values. To solve this, incentives were introduced: the oracle also votes retrospectively on the price one week ago, and if your (real time *or* retrospective) vote is more than 1% away from the median retrospective vote, you are heavily penalized, with the penalty going to those who voted "correctly".

Within a year, over 90% of validators are participating. Someone asked: what if Lido bands together with a few other large stakers to 51% attack the vote, forcing through a fake ETH/USD price value, extracting heavy penalties from everyone who does not participate in the attack? The oracle's proponents, at this point heavily invested in the scheme, reply: well if that happens, Ethereum will surely fork to kick the bad guys out.

At first, the scheme is limited to ETH/USD, and it appears resilient and stable. But over the years, other indices get added: ETH/EUR, ETH/CNY, and eventually rates for all countries in the G20.

But in 2034, things start to go wrong. Brazil has an unexpectedly severe political crisis, leading to a disputed election. One political party ends up in control of the capital and 75% of the country, but another party ends up in control of some northern areas. Major Western media argue that the northern party is clearly the legitimate winner because it acted legally and the southern party acted illegally (and by the way are fascist). Indian and Chinese official sources, and Elon Musk, argue that the southern party has actual control of most of the country, and the international community should not try to be a world police and should instead accept the outcome.

By this point, Brazil has a CBDC, which splits into two forks: the (northern) BRL-N, and the (southern) BRL-S. When voting in the oracle, 60% of Ethereum stakers provide the ETH/BRL-S rate. Major community leaders and businesses decry the stakers' craven capitulation to fascism, and propose to fork the chain to only include the "good stakers" providing the ETH/BRL-N rate, and drain

the other stakers' balances to near-zero. Within their social media bubble, they believe that they will clearly win. However, once the fork hits, the BRL-S side proves unexpectedly strong. What they expected to be a landslide instead proves to be pretty much a 50-50 community split.

At this point, the two sides are in their two separate universes with their two chains, with no practical way of coming back together. Ethereum, a global permissionless platform created in part to be a refuge from nations and geopolitics, instead ends up cleaved in half by any one of the twenty G20 member states having an unexpectedly severe internal issue.

That's a nice scifi story you got there. Could even make a good movie. But what can we actually learn from it?

A blockchain's "purity", in the sense of it being a purely mathematical construct that attempts to come to consensus only on purely mathematical things, is a huge advantage. As soon as a blockchain tries to "hook in" to the outside world, the outside world's conflicts start to impact on the blockchain too. Given a sufficiently extreme political event - in fact, not *that* extreme a political event, given that the above story was basically a pastiche of events that have actually happened in various major (>25m population) countries all within the past decade - even something as benign as a currency oracle could tear the community apart.

Here are a few more possible scenarios:

- One of the currencies that the oracle tracks (which could even be USD) simply hyperinflates, and markets break down to the point that at some points in time there is no clear specific market price.
- If Ethereum adds a price oracle to *another cryptocurrency*, then a controversial split like in the story above is not hypothetical: it's something that has already happened, including in the histories of both [Bitcoin](#) and [Ethereum itself](#).
- If strict capital controls become operational, then *which* price to report as the legitimate market price between two currencies becomes a political question.

But more importantly, I'd argue that there is a Schelling fence at play: once a blockchain *starts* incorporating real-world price indices as a layer-1 protocol feature, it could easily succumb to interpreting more and more real-world information. Introducing layer-1 price indices also expands the blockchain's legal attack surface: instead of being *just* a neutral technical platform, it becomes much more explicitly a financial tool.

What about risks from examples other than price indices?

Any expansion of the "duties" of Ethereum's consensus increases the costs, complexities and risks of running a validator. Validators become required to take on the human effort of paying attention and running and updating additional software to make sure that they are acting correctly according to whatever other protocols are being introduced. Other communities gain the ability to externalize their dispute resolution needs onto the Ethereum community. Validators and the Ethereum community as a whole become forced to make far more decisions, each of which has some risk of causing a community split. Even if there is no split, the desire to avoid such pressure creates additional incentives to externalize the decisions to centralized entities through stake-pooling.

The possibility of a split would also greatly strengthen perverse too-big-to-fail mechanics. There are so many layer-2 and application-layer projects on Ethereum that it would be impractical for Ethereum social consensus to be willing to fork to solve *all* of their problems. Hence, larger projects would inevitably get a larger chance of getting a bailout than smaller ones. This would in turn lead to larger projects getting a moat: would you rather have your coins on Arbitrum or Optimism, where if something goes wrong Ethereum will fork to save the day, or on [Taiko](#), where because it's smaller (and non-Western, hence less socially connected to core dev circles), an L1-backed rescue is much less likely?

But bugs are a risk, and we need better oracles. So what should we do?

The best solutions to these problems are, in my view, case-by-case, because the various problems are inherently so different from each other. Some solutions include:

- **Price oracles**: either [not-quite-cryptoeconomic decentralized oracles](#), or validator-voting-based

oracles that **explicitly commit to their emergency recovery strategies being something other than appealing to L1 consensus** for recovery (or some combination of both). For example, a price oracle could count on a trust assumption that voting participants get corrupted slowly, and so users would have early warning of an attack and could exit any systems that depend on the oracle. Such an oracle could intentionally give its reward only after a long delay, so that if that instance of the protocol falls into disuse (eg. because the oracle fails and the community forks toward another version), the participants do not get the reward.

- **More complex truth oracles** reporting on facts more subjective than price: some kind of decentralized court system built on a [not-quite-cryptoeconomic DAO](#).
- **Layer 2 protocols**:
  - In the short term, rely on partial training wheels (what this post calls [stage 1](#))
  - In the medium term, rely on [multiple proving systems](#). Trusted hardware (eg. SGX) could be included here; I strongly anti-endorse SGX-like systems as a *sole* guarantor of security, but as a member of a 2-of-3 system they could be valuable.
  - In the longer term, hopefully complex functionalities such as "EVM validation" will themselves eventually be enshrined in the protocol
- **Cross-chain bridges**: similar logic as oracles, but also, try to minimize how much you rely on bridges at all: hold assets on the chain where they originate and use atomic swap protocols to move value between different chains.
- **Using the Ethereum validator set to secure other chains**: one reason why the (safer) Dogecoin approach in the [list of examples above](#) might be insufficient is that while it does protect against 51% *finality-reversion* attacks, it does not protect against 51% *censorship* attacks. However, if you are already relying on Ethereum validators, then one possible direction to take is to move away from trying to manage an independent chain entirely, and become a [validium](#) with proofs anchored into Ethereum. If a chain does this, its protection against finality-reversion attacks becomes as strong as Ethereum's, and it becomes secure against censorship up to 99% attacks (as opposed to 49%).

Conclusions

Blockchain communities' social consensus is a fragile thing. It's necessary - because upgrades happen, bugs happen, and 51% attacks are always a possibility - but because it has such a high risk of causing chain splits, in mature communities it should be used sparingly. There is a natural urge to try to extend the blockchain's core with more and more functionality, because the blockchain's core has the largest economic weight and the largest community watching it, but each such extention makes the core itself more fragile.

We should be wary of application-layer projects taking actions that risk increasing the "scope" of blockchain consensus to anything other than verifying the core Ethereum protocol rules. It is natural for application-layer projects to attempt such a strategy, and indeed such ideas are often simply conceived without appreciation of the risks, but its result can easily become very misaligned with the goals of the community as a whole. Such a process has no limiting principle, and could easily lead to a blockchain community having more and more "mandates" over time, pushing it into an uncomfortable choice between a high yearly risk of splitting and some kind of de-facto formalized bureaucracy that has ultimate control of the chain.

We should instead preserve the chain's minimalism, support uses of re-staking that do not look like slippery slopes to extending the role of Ethereum consensus, and help developers find alternate strategies to achieve their security goals.

Travel time ~= 750 * distance ^ 0.6
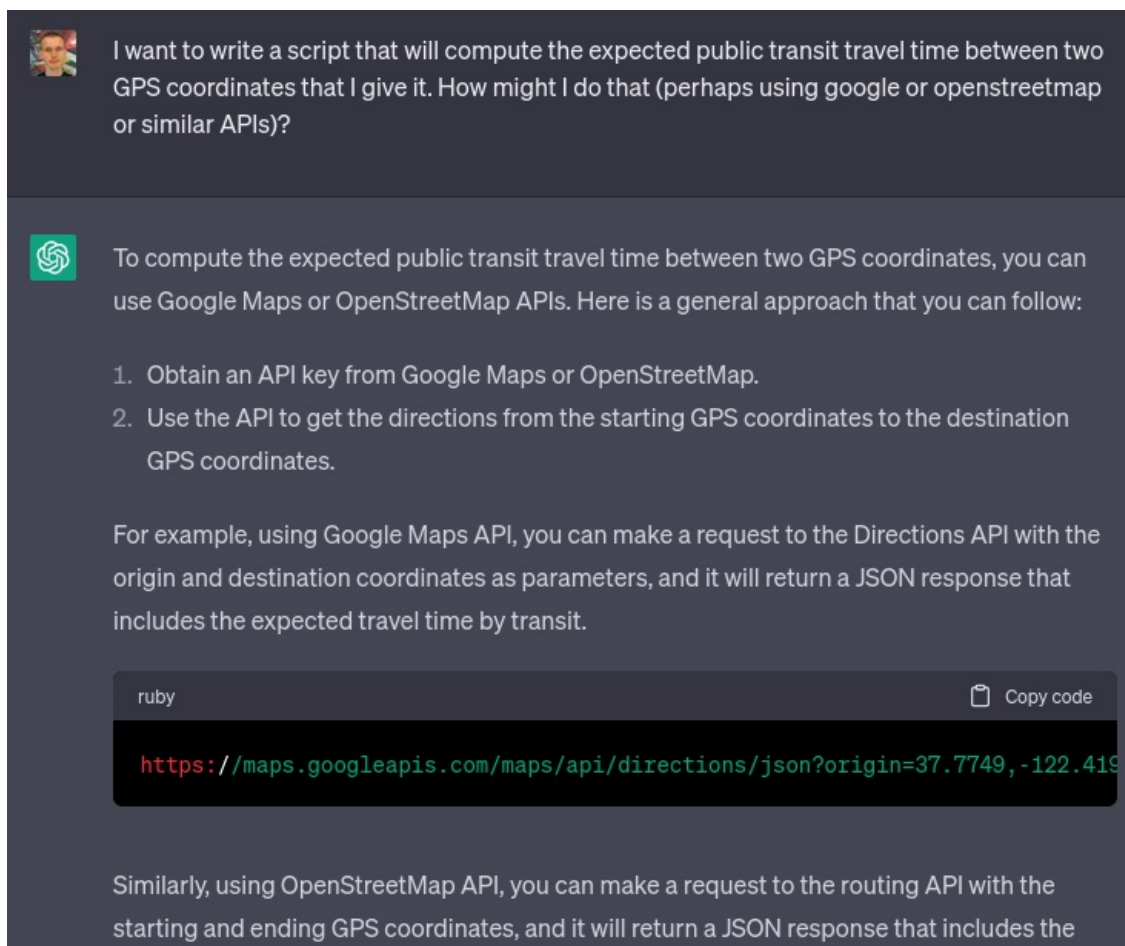
2023 Apr 14

As another exercise in using ChatGPT 3.5 to do weird things and seeing what happens, I decided to explore an interesting question: how does the time it takes to travel from point A to point B scale with distance, *in the real world*? That is to say, if you sample randomly from positions where people are actually at (so, for example, 56% of points you choose would be in cities), and you use public transportation, how does travel time scale with distance?

Obviously, travel time would grow slower than linearly: the further you have to go, the more opportunity you have to resort to forms of transportation that are faster, but have some fixed overhead. Outside of a very few lucky cases, there is no practical way to take a bus to go faster if your destination is 170 meters away, but if your destination is 170 *kilometers* away, you suddenly get more options. And if it's 1700 kilometers away, you get airplanes.

So I asked ChatGPT for the ingredients I would need:

I went with the [GeoLife dataset](). I did notice that while it claims to be about users around the world,

primarily it seems to focus on people in Seattle and Beijing, though they do occasionally visit other cities. That said, I'm not a perfectionist and I was fine with it. I asked ChatGPT to write me a script to interpret the dataset and extract a randomly selected coordinate from each file:
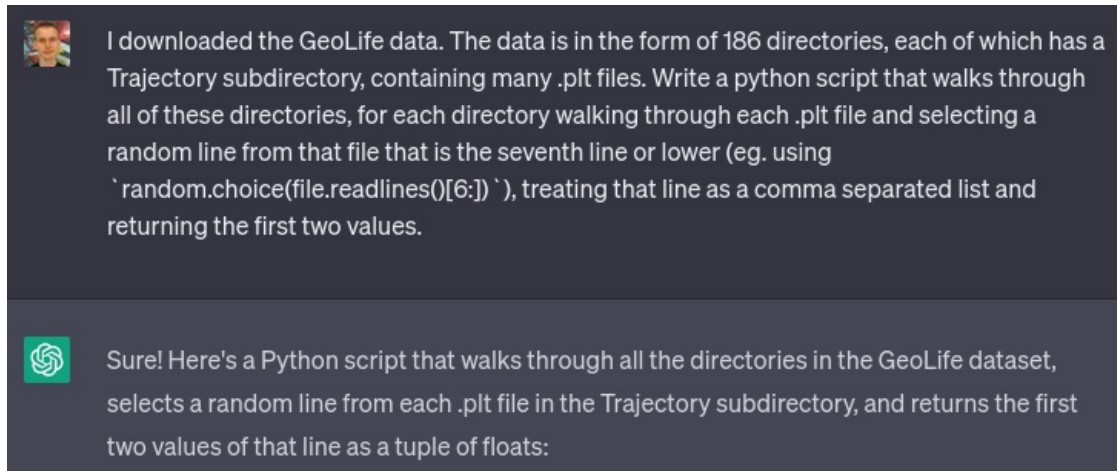
Amazingly, it *almost* succeeded on the first try. It did make the mistake of assuming every item in the list would be a number (`values = [float(x) for x in line.strip().split(',')]`), though perhaps to some extent that was my fault: when I said "the first two *values*" it probably interpreted that as implying that the entire line was made up of "values" (ie. numbers).

I fixed the bug manually. Now, I have a way to get some randomly selected points where people are at, and I have an API to get the public transit travel time between the points.

I asked it for more coding help:

- Asking how to get an API key for the Google Maps Directions API (it gave an answer that seems to be outdated, but that succeeded at immediately pointing me to the right place)
- Writing a function to compute the straight-line distance between two GPS coordinates (it gave the correct answer on the first try)
- Given a list of (`distance`, `time`) pairs, drawing a scatter plot, with time and distance as axes, both axes logarithmically scaled (it gave the correct answer on the first try)
- Doing a linear regression on the logarithms of distance and time to try to fit the data to a power law (it bugged on the first try, succeeded on the second)

This gave me some really nice data (this is filtered for distances under 500km, as above 500km the best path almost certainly includes flying, and the Google Maps directions don't take into account flights):
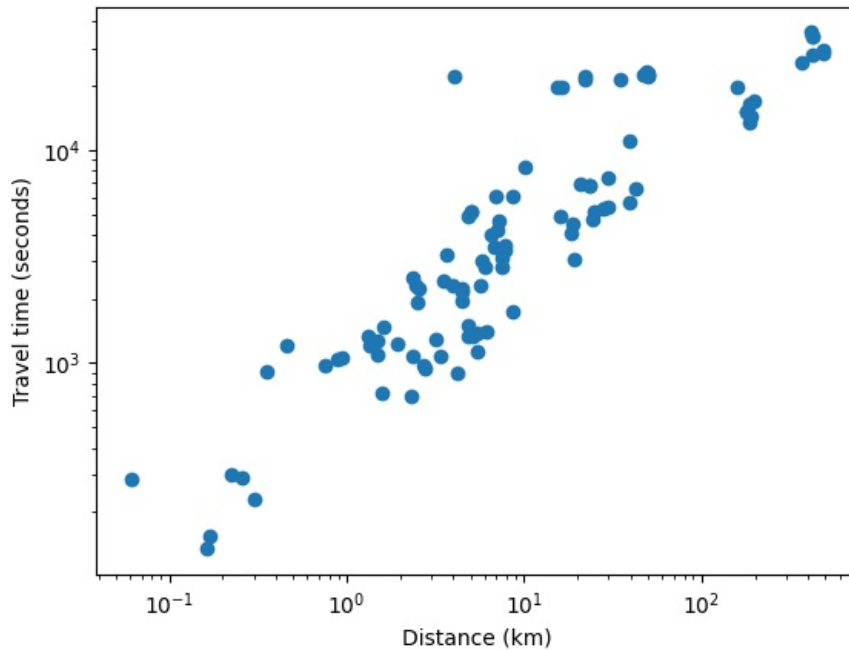
The power law fit that the linear regression gave is: `travel_time = 965.8020738916074 * distance^0.6138556361612214` (time in seconds, distance in km).

Now, I needed travel time data for longer distances, where the optimal route would include flights. Here, APIs could not help me: I asked ChatGPT if there were APIs that could do such a thing, and it did not give a satisfactory answer. I resorted to doing it manually:

- I used the same script, but modified it slightly to only output pairs of points which were *more* than 500km apart from each other.
- I took the first 8 results within the United States, and the first 8 with at least one end outside the United States, skipping over results that represented a city pair that had already been covered.
- For each result I manually obtained:
  - `to_airport`: the public transit travel time from the starting point to the nearest airport, using Google Maps outside China and [Baidu Maps](#) inside China.
  - `from_airport`: the public transit travel time to the end point from the nearest airport
  - `flight_time`: the flight time from the starting point to the end point. I used [Google Flights](#)) and always took the top result, except in cases where the top result was completely crazy (more than 2x the length of the shortest), in which case I took the shortest.
- I computed the travel time as `(to_airport) * 1.5 + (90 if international else 60) + flight_time + from_airport`. The first part is a fairly aggressive formula (I personally am much more conservative than this) for when to leave for the airport: aim to arrive 60 min before if domestic and 90 min before if international, and multiply expected travel time by 1.5x in case there are any mishaps or delays.

This was boring and I was not interested in wasting my time to do more than 16 of these; I presume if I was a serious researcher I would already have an account set up on [TaskRabbit](#) or some similar service that would make it easier to hire other people to do this for me and get much more data. In any case, 16 is enough; I put my resulting data [here](#).

Finally, just for fun, I added some data for how long it would take to travel to various locations in space: [the moon](#) (I added 12 hours to the time to take into account an average person's travel time to the launch site), [Mars](#), [Pluto](#) and [Alpha](#) [Centauri](#). You can find my complete code [here](#).

Here's the resulting chart:
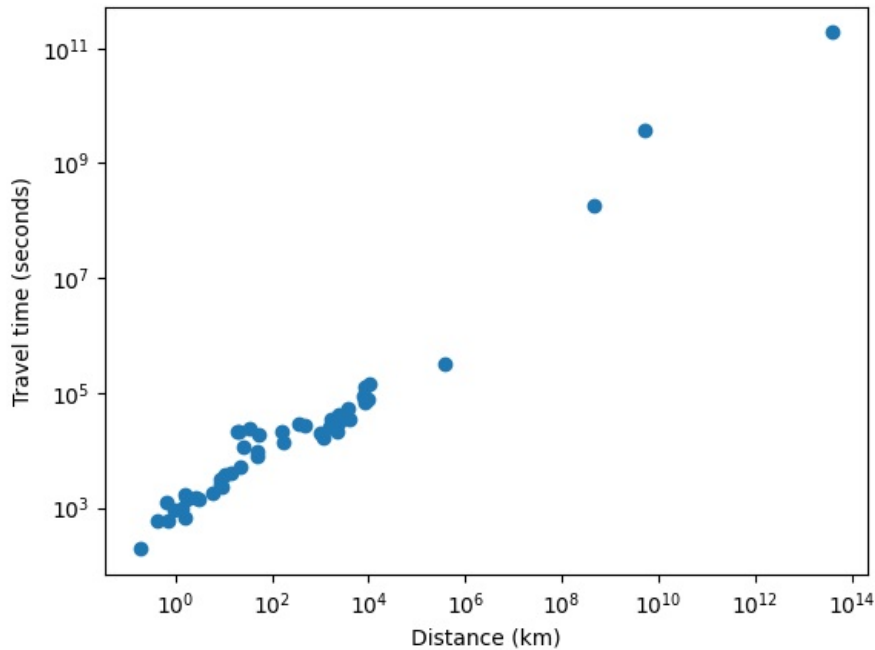
```
travel_time = 733.002223593754 * distance^0.591980777827876
```

WAAAAAT?!?!! From this chart it seems like there is a surprisingly precise relationship governing travel time from point A to point B that somehow holds across such radically different transit media as walking, subways and buses, airplanes and (!!) interplanetary and hypothetical interstellar spacecraft. I swear that I am not cherrypicking; I did not throw out any data that was inconvenient, everything (including the space stuff) that I checked I put on the chart.

ChatGPT 3.5 worked impressively well this time; it certainly stumbled and fell *much* less than my previous misadventure, where I tried to get it to help me convert IPFS bafyhashes into hex. In general, ChatGPT seems uniquely good at teaching me about libraries and APIs I've never heard of before but that other people use all the time; this reduces the barrier to entry between amateurs and professionals which seems like a very positive thing.

So there we go, there seems to be some kind of really weird fractal law of travel time. Of course, different transit technologies could change this relationship: if you replace public transit with cars and commercial flights with private jets, travel time becomes somewhat more linear. And once we upload our minds onto computer hardware, we'll be able to travel to Alpha Centauri on much crazier vehicles like ultralight craft propelled by Earth-based lightsails) that could let us go anywhere at a significant fraction of the speed of light. But for now, it does seem like there is a strangely consistent relationship that puts time much closer to the square root of distance.

How will Ethereum's multi-client philosophy interact with ZK-EVMs?

2023 Mar 31

Special thanks to Justin Drake for feedback and review

One underdiscussed, but nevertheless very important, way in which Ethereum maintains its security and decentralization is its **multi-client philosophy**. Ethereum intentionally has no "reference client" that everyone runs by default: instead, there is a collaboratively-managed **specification** (these days written in the very human-readable but very slow Python) and there are multiple teams making **implementations** of the spec (also called "**clients**"), which is what users actually run.
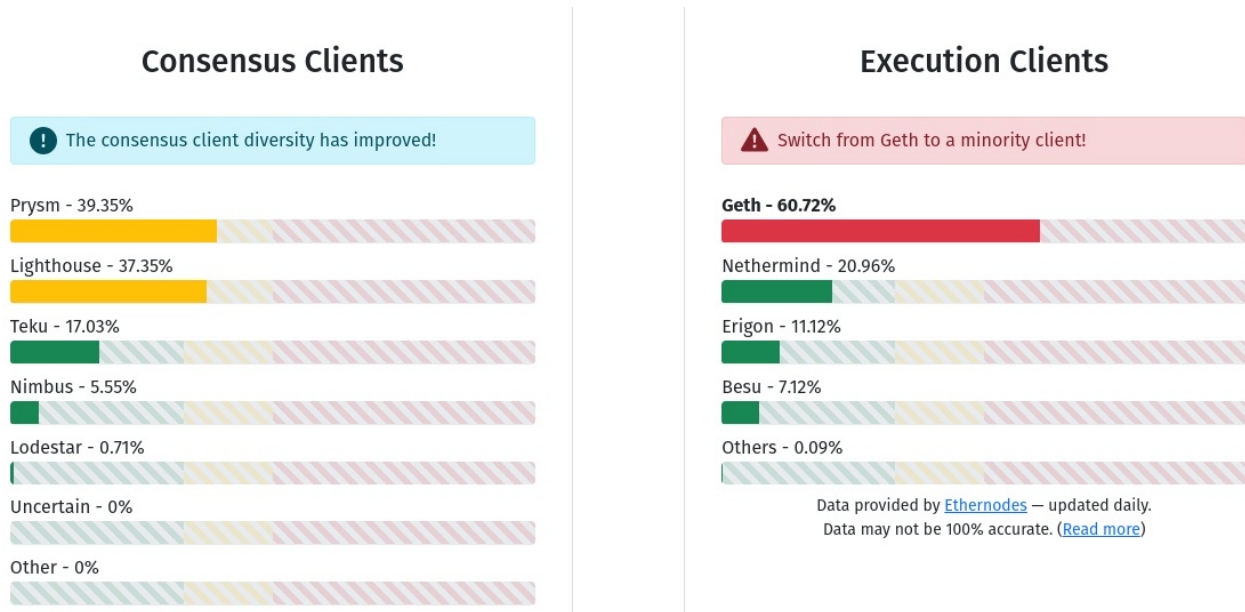
Each Ethereum node runs a consensus client and an execution client. As of today, no consensus or execution client makes up more than 2/3 of the network. If a client with less than 1/3 share in its category has a bug, the network would simply continue as normal. If a client with between 1/3 and 2/3 share in its category (so, Prysm, Lighthouse or Geth) has a bug, the chain would continue adding blocks, but it would stop finalizing blocks, giving time for developers to intervene.

One underdiscussed, but nevertheless very important, major upcoming transition in the way the Ethereum chain gets validated is the rise of **ZK-EVMs**. SNARKs proving EVM execution have been under development for years already, and the technology is actively being used by layer 2 protocols called ZK rollups. Some of these ZK rollups are active on mainnet today, with more coming soon. But in the longer term, ZK-EVMs are not just going to be for rollups; we want to use them to verify execution on layer 1 as well (see also: the Verge).

Once that happens, ZK-EVMs de-facto become a third type of Ethereum client, just as important to the network's security as execution clients and consensus clients are today. And this naturally raises a question: how will ZK-EVMs interact with the multi-client philosophy? One of the hard parts is already done: we already have multiple ZK-EVM implementations that are being actively developed. But other hard parts remain: how would we actually make a "multi-client" ecosystem for ZK-proving correctness of Ethereum blocks? This question opens up some interesting technical challenges - and of course the looming question of whether or not the tradeoffs are worth it.

What was the original motivation for Ethereum's multi-client philosophy?

Ethereum's multi-client philosophy is a type of decentralization, and like decentralization in general, one can focus on either the technical benefits of architectural decentralization or the social benefits of political decentralization. Ultimately, the multi-client philosophy was motivated by both and serves both.

Arguments for technical decentralization

The main benefit of technical decentralization is simple: it reduces the risk that one bug in one piece of software leads to a catastrophic breakdown of the entire network. A historical situation that exemplifies this risk is the 2010 Bitcoin overflow bug. At the time, the Bitcoin client code did not check that the sum of the outputs of a transaction does not overflow (wrap around to zero by summing to above the maximum integer of $2^{64} - 1$), and so someone made a transaction that did exactly that, giving themselves billions of bitcoins. The bug was discovered within hours, and a fix was rushed through and quickly deployed across the network, but had there been a mature ecosystem at the time, those coins would have been accepted by exchanges, bridges and other structures, and the attackers could have gotten away with a lot of money. If there had been five different Bitcoin clients, it would have been very unlikely that all of them had the same bug, and so there would have been an immediate split, and the side of the split that was buggy would have probably lost.

There is a tradeoff in using the multi-client approach to minimize the risk of catastrophic bugs: instead, you get **consensus failure** bugs. That is, if you have two clients, there is a risk that the clients have subtly different interpretations of some protocol rule, and while both interpretations are reasonable and do not allow stealing money, the disagreement would cause the chain to split in half. A serious split of that type happened once in Ethereum's history (there have been other much smaller splits where very small portions of the network running old versions of the code forked off). Defenders of the single-client approach point to consensus failures as a reason to not have multiple implementations: if there is only one client, that one client will not disagree with itself. Their model of how number of clients

translates into risk might look something like this:
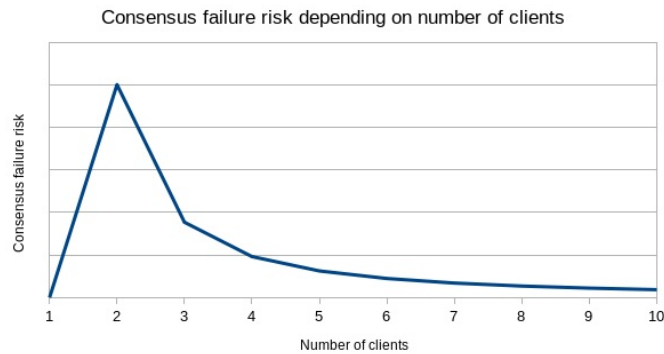
I, of course, disagree with this analysis. The crux of my disagreement is that (i) 2010-style catastrophic bugs matter too, and (ii) **you never *actually* have only one client**. The latter point is made most obvious by the [Bitcoin fork of 2013](#): a chain split occurred because of a disagreement between *two different versions* of the Bitcoin client, one of which turned out to have an accidental and undocumented limit on the number of objects that could be modified in a single block. Hence, one client in theory is often two clients in practice, and five clients in theory might be six or seven clients in practice - so we should just take the plunge and go on the right side of the risk curve, and have at least a few different clients.

Arguments for political decentralization

Monopoly client developers are in a position with a lot of political power. If a client developer proposes a change, and users disagree, *theoretically* they could refuse to download the updated version, or create a fork without it, but *in practice* it's often difficult for users to do that. What if a disagreeable protocol change is bundled with a necessary security update? What if the main team threatens to quit if they don't get their way? Or, more tamely, what if the monopoly client team ends up being the only group with the greatest protocol expertise, leaving the rest of the ecosystem in a poor position to judge technical arguments that the client team puts forward, leaving the client team with a lot of room to push their own particular goals and values, which might not match with the broader community?

Concern about protocol politics, particularly in the context of the [2013-14 Bitcoin OP_RETURN wars](#) where some participants were openly in favor of discriminating against particular usages of the chain, was a significant contributing factor in Ethereum's early adoption of a multi-client philosophy, which was aimed to make it harder for a small group to make those kinds of decisions. Concerns specific to the Ethereum ecosystem - namely, avoiding concentration of power within the Ethereum Foundation itself - provided further support for this direction. In 2018, a decision was made to intentionally have the Foundation *not* make an implementation of the Ethereum PoS protocol (ie. what is now called a "consensus client"), leaving that task entirely to outside teams.

How will ZK-EVMs come in on layer 1 in the future?

Today, ZK-EVMs are used in rollups. This increases scaling by allowing expensive EVM execution to happen only a few times off-chain, with everyone else simply verifying [SNARKs](#) posted on-chain that prove that the EVM execution was computed correctly. It also allows some data (particularly signatures) to not be included on-chain, saving on gas costs. This gives us a lot of scalability benefits, and the combination of scalable computation with ZK-EVMs and scalable data with [data availability sampling](#) could let us scale very far.

However, the Ethereum network today also has a different problem, one that no amount of layer 2 scaling can solve by itself: the layer 1 is difficult to verify, to the point where not many users run their own node. Instead, most users simply trust third-party providers. Light clients such as [Helios](#) and [Succinct](#) are taking steps toward solving the problem, but a light client is far from a fully verifying node: a light client merely verifies the signatures of a random subset of validators called the [sync committee](#), and does not verify that the chain actually follows the protocol rules. To bring us to a world where users can actually verify that the chain follows the rules, we would have to do something different.

Option 1: constrict layer 1, force almost all activity to move to layer 2

We could over time reduce the layer 1 gas-per-block target down from 15 million to 1 million, enough for a block to contain a single SNARK and a few deposit and withdraw operations but not much else, and thereby force almost all user activity to move to layer 2 protocols. Such a design could still support many rollups committing in each block: we could use [off-chain aggregation protocols](#) run by customized builders to gather together SNARKs from multiple layer 2 protocols and combine them into a single SNARK. **In such a world, the *only* function of layer 1 would be to be a clearinghouse for layer 2 protocols, verifying their proofs and occasionally facilitating large funds transfers between them**.
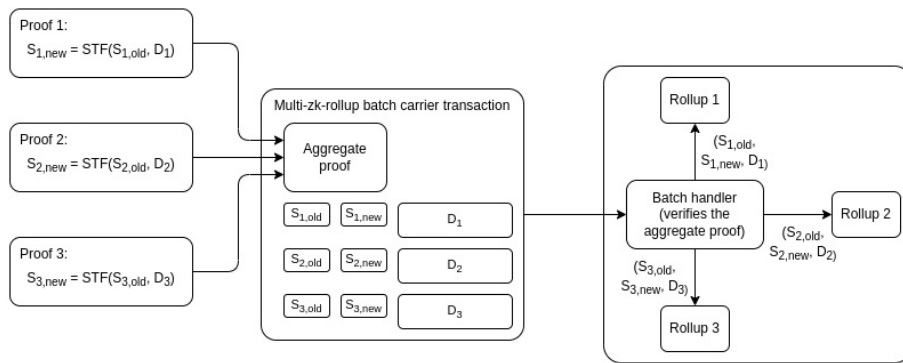
This approach could work, but it has several important weaknesses:

- **It's de-facto backwards-incompatible**, in the sense that many existing L1-based applications become economically nonviable. User funds up to hundreds or thousands of dollars could get stuck as fees become so high that they exceed the cost of emptying those accounts. This could be addressed by letting users sign messages to opt in to an in-protocol mass migration to an L2 of their choice (see some early implementation ideas here), but this adds complexity to the transition, and making it truly cheap enough would require *some* kind of SNARK at layer 1 anyway. I'm generally a fan of breaking backwards compatibility when it comes to things like the SELFDESTRUCT opcode, but in this case the tradeoff seems much less favorable.
- **It might still not make verification cheap enough**. Ideally, the Ethereum protocol should be easy to verify not just on laptops but also inside phones, browser extensions, and even inside other chains. Syncing the chain for the first time, or after a long time offline, should also be easy. A laptop node could verify 1 million gas in ~20 ms, but even that implies 54 seconds to sync after one day offline (assuming single slot finality increases slot times to 32s), and for phones or browser extensions it would take a few hundred milliseconds per block and might still be a non-negligible battery drain. These numbers are manageable, but they are not ideal.
- **Even in an L2-first ecosystem, there are benefits to L1 being at least somewhat affordable**. Validiums can benefit from a stronger security model if users can withdraw their funds if they notice that new state data is no longer being made available. Arbitrage becomes more efficient, especially for smaller tokens, if the minimum size of an economically viable cross-L2 direct transfer is smaller.

Hence, it seems more reasonable to try to find a way to use ZK-SNARKs to verify the layer 1 itself.

Option 2: SNARK-verify the layer 1

A type 1 (fully Ethereum-equivalent) ZK-EVM can be used to verify the EVM execution of a (layer 1) Ethereum block. We could write more SNARK code to also verify the consensus side of a block. This would be a challenging engineering problem: today, ZK-EVMs take minutes to hours to verify Ethereum blocks, and generating proofs in real time would require one or more of (i) improvements to Ethereum itself to remove SNARK-unfriendly components, (ii) either large efficiency gains with specialized hardware, and (iii) architectural improvements with much more parallelization. However, there is no fundamental technological reason why it cannot be done - and so I expect that, even if it takes many years, it will be done.

Here is where we see the intersection with the multi-client paradigm: if we use ZK-EVMs to verify layer 1, which ZK-EVM do we use?

I see three options:

1. **Single ZK-EVM**: abandon the multi-client paradigm, and choose a single ZK-EVM that we use to verify blocks.
2. **Closed multi ZK-EVM**: agree on and enshrine in consensus a specific set of multiple ZK-EVMs, and have a consensus-layer protocol rule that a block needs proofs from more than half of the ZK-EVMs in that set to be considered valid.
3. **Open multi ZK-EVM**: different clients have different ZK-EVM implementations, and each client waits for a proof that is compatible with its own implementation before accepting a block as valid.

To me, (3) seems ideal, at least until and unless our technology improves to the point where we can formally prove that all of the ZK-EVM implementations are equivalent to each other, at which point we can just pick whichever one is most efficient. (1) would sacrifice the benefits of the multi-client paradigm, and (2) would close off the possibility of developing new clients and lead to a more centralized ecosystem. (3) has challenges, but those challenges seem smaller than the challenges of the other two options, at least for now.

Implementing (3) would not be too hard: one might have a p2p sub-network for each type of proof, and a client that uses one type of proof would listen on the corresponding sub-network and wait until they receive a proof that their verifier recognizes as valid.

The two main challenges of (3) are likely the following:

- **The latency challenge**: a malicious attacker could publish a block late, along with a proof valid for one client. It would realistically take a long time (even if eg. 15 seconds) to generate proofs valid for other clients. This time would be long enough to potentially create a temporary fork and disrupt the chain for a few slots.
- **Data inefficiency**: one benefit of ZK-SNARKs is that data that is only relevant to *verification* (sometimes called "witness data") could be removed from a block. For example, once you've verified a signature, you don't need to keep the signature in a block, you could just store a single bit saying that the signature is valid, along with a single proof in the block confirming that all of the valid signatures exist. However, if we want it to be possible to generate proofs of multiple types for a block, the original signatures would need to actually be published.

The latency challenge could be addressed by being careful when designing the single-slot finality protocol. Single-slot finality protocols will likely require more than two rounds of consensus per slot, and so one could require the first round to include the block, and only require nodes to verify proofs before signing in the third (or final) round. This ensures that a significant time window is always available between the deadline for publishing a block and the time when it's expected for proofs to be available.

The data efficiency issue would have to be addressed by having a separate protocol for aggregating verification-related data. For signatures, we could use BLS aggregation, which ERC-4337 already supports. Another major category of verification-related data is ZK-SNARKs used for privacy. Fortunately, these often tend to have their own aggregation protocols.

It is also worth mentioning that SNARK-verifying the layer 1 has an important *benefit*: the fact that on-chain EVM execution no longer needs to be verified by every node makes it possible to greatly increase the amount of EVM execution taking place. This could happen

either by greatly increasing the layer 1 gas limit, or by introducing enshrined rollups, or both.

Conclusions

Making an open multi-client ZK-EVM ecosystem work well will take a lot of work. But the really good news is that much of this work is happening or will happen anyway:

- We have multiple strong ZK-EVM implementations already. These implementations are not yet type 1 (fully Ethereum-equivalent), but many of them are actively moving in that direction.
- The work on light clients such as Helios and Succinct may eventually turn into a more full SNARK-verification of the PoS consensus side of the Ethereum chain.
- Clients will likely start experimenting with ZK-EVMs to prove Ethereum block execution on their own, especially once we have stateless clients and there's no technical need to directly re-execute every block to maintain the state. We will probably get a slow and gradual transition from clients verifying Ethereum blocks by re-executing them to most clients verifying Ethereum blocks by checking SNARK proofs.
- The ERC-4337 and PBS ecosystems are likely to start working with aggregation technologies like BLS and proof aggregation pretty soon, in order to save on gas costs. On BLS aggregation, work has already started.

With these technologies in place, the future looks very good. Ethereum blocks would be smaller than today, anyone could run a fully verifying node on their laptop or even their phone or inside a browser extension, and this would all happen while preserving the benefits of Ethereum's multi-client philosophy.

In the longer-term future, of course anything could happen. Perhaps AI will super-charge formal verification to the point where it can easily prove ZK-EVM implementations equivalent and identify all the bugs that cause differences between them. Such a project may even be something that could be practical to start working on now. If such a formal verification-based approach succeeds, different mechanisms would need to be put in place to ensure continued political decentralization of the protocol; perhaps at that point, the protocol would be considered "complete" and immutability norms would be stronger. But even if that is the longer-term future, the open multi-client ZK-EVM world seems like a natural stepping stone that is likely to happen anyway.

In the nearer term, this is still a long journey. ZK-EVMs *are* here, but ZK-EVMs becoming truly viable at layer 1 would require them to become type 1, and make proving fast enough that it can happen in real time. With enough parallelization, this is doable, but it will still be a lot of work to get there. Consensus changes like raising the gas cost of KECCAK, SHA256 and other hash function precompiles will also be an important part of the picture. That said, the first steps of the transition may happen sooner than we expect: once we switch to Verkle trees and stateless clients, clients could start gradually using ZK-EVMs, and a transition to an "open multi-ZK-EVM" world could start happening all on its own.

Some personal user experiences

2023 Feb 28

In 2013, I went to a sushi restaurant beside the Internet Archive in San Francisco, because I had heard that it accepted bitcoin for payments and I wanted to try it out. When it came time to pay the bill, I asked to pay in BTC. I scanned the QR code, and clicked "send". To my surprise, the transaction did not go through; it appeared to have been sent, but the restaurant was not receiving it. I tried again, still no luck. I soon figured out that the problem was that my mobile internet was not working well at the time. I had to walk over 50 meters toward the Internet Archive nearby to access its wifi, which finally allowed me to send the transaction.

Lesson learned: internet is not 100% reliable, and customer internet is less reliable than merchant internet. We need in-person payment systems to have *some* functionality (NFC, customer shows a QR code, whatever) to allow customers to transfer their transaction data directly to the merchant if that's the best way to get it broadcasted.

In 2021, I attempted to pay for tea for myself and my friends at a coffee shop in Argentina. In their defense, they did not *intentionally* accept cryptocurrency: the owner simply recognized me, and showed me that he had an account at a cryptocurrency exchange, so I suggested to pay in ETH (using cryptocurrency exchange accounts as wallets is a standard way to do in-person payments in Latin America). Unfortunately, my first transaction of 0.003 ETH did not get accepted, probably because it was under the exchange's 0.01 ETH deposit minimum. I sent another 0.007 ETH. Soon, both got confirmed. (I did not mind the 3x overpayment and treated it as a tip).

In 2022, I attempted to pay for tea at a different location. The first transaction failed, because the default transaction from my mobile wallet sent with only 21000 gas, and the receiving account was a contract that required extra gas to process the transfer. Attempts to send a second transaction failed, because a UI glitch in my phone wallet made it not possible to scroll down and edit the field that contained the gas limit.

Lesson learned: simple-and-robust UIs are better than fancy-and-sleek ones. But also, most users don't even know what gas limits are, so we really just need to have better defaults.

Many times, there has been a surprisingly long time delay between my transaction getting accepted on-chain, and the service acknowledging the transaction, even as "unconfirmed". Some of those times, I definitely got worried that there was some glitch with the payment system on their side.

Many times, there has been a surprisingly long and unpredictable time delay between sending a transaction, and that transaction getting accepted in a block. Sometimes, a transaction would get accepted in a few seconds, but other times, it would take minutes or even hours. Recently, EIP-1559 significantly improved this, ensuring that most transactions get accepted into the next block, and even more recently the Merge improved things further by stabilizing block times.
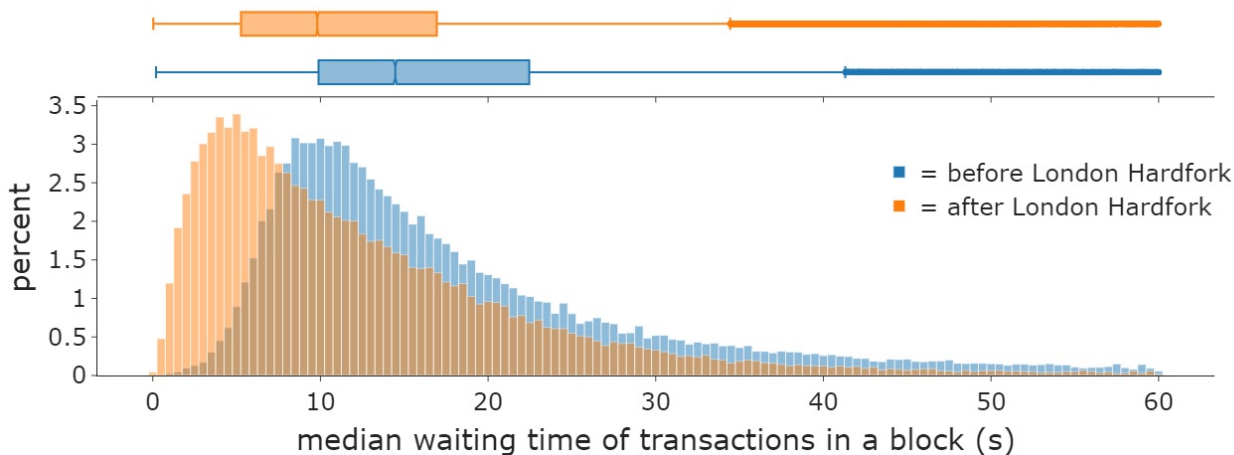
Diagram from this report by Yinhong (William) Zhao and Kartik Nayak.

However, outliers still remain. If you send a transaction at the same time as when many others are sending transactions and the base fee is spiking up, you risk the base fee going too high and your transaction not getting accepted. Even worse, wallet UIs suck at showing this. There are no big red flashing alerts, and very little clear indication of what you're supposed to do to solve this problem. Even to an expert, who knows that in this case you're supposed to "speed up" the transaction by publishing a new transaction with identical data but a higher max-basefee, it's often not clear where the button to do that actually is.

Lesson learned: UX around transaction inclusion needs to be improved, though there are fairly simple fixes. Credit to the Brave wallet team for taking my suggestions on this topic seriously, and first increasing the max-basefee tolerance from 12.5% to 33%, and more recently exploring ways to make stuck transactions more obvious in the UI.

In 2019, I was testing out one of the earliest wallets that was attempting to provide social recovery. Unlike my preferred approach, which is smart-contract-based, their approach was to use Shamir's secret sharing to split up the private key to the account into five pieces, in such a way that any three of those pieces could be used to recover the private key. Users were expected to choose five friends ("**guardians**" in modern lingo), convince them to download a separate mobile application, and provide a confirmation code that would be used to create an encrypted connection from the user's wallet to the friend's application through Firebase and send them their share of the key.

This approach quickly ran into problems for me. A few months later, something happened to my wallet and I needed to actually use the recovery procedure to recover it. I asked my friends to perform the recovery procedure with me through their apps - but it did not go as planned. Two of them lost their key shards, because they switched phones and forgot to move the recovery application over. For a third, the Firebase connection mechanism did not work for a long time. Eventually, we figured out how to fix the issue, and recover the key. A few months after that, however, the wallet broke again. This time, a regular software update somehow accidentally reset the app's storage and deleted its key. But I had not added enough recovery partners, because the Firebase connection mechanism was too broken and was not letting me successfully do that. I ended up losing a small amount of BTC and ETH.

Lesson learned: secret-sharing-based off-chain social recovery is just really fragile and a bad idea unless there are no other options. Your recovery guardians should not have to download a separate application, because if you have an application *only* for an exceptional situation like recovery, it's too easy to forget about it and lose it. Additionally, requiring separate centralized communication channels comes with all kinds of problems. Instead, the way to add guardians should be to provide their ETH address, and recovery should be done by smart contract, using ERC-4337 account abstraction wallets. This way, the guardians would only need to not lose their Ethereum wallets, which is something that they already care much more about not losing for other reasons.

In 2021, I was attempting to save on fees when using Tornado Cash, by using the "self-relay" option. Tornado Cash uses a "relay" mechanism where a third party pushes the transaction on-chain, because when you are withdrawing you generally do not yet have coins in your withdrawal address, and you don't want to pay for the transaction with your deposit address because that creates a public link between the two addresses, which is the whole problem that Tornado Cash is trying to prevent. The problem is that the relay mechanism is often expensive, with relays charging a percentage fee that could go far above the

actual gas fee of the transaction.

To save costs, one time I used the relay for a first small withdrawal that would charge lower fees, and then used the "self-relay" feature in Tornado Cash to send a second larger withdrawal myself without using relays. The problem is, I screwed up and accidentally did this while logged in to my deposit address, so the deposit address paid the fee instead of the withdrawal address. Oops, I created a public link between the two.

Lesson learned: wallet developers should start thinking much more explicitly about privacy. Also, we need better forms of account abstraction to remove the need for centralized or even federated relays, and commoditize the relaying role.

Miscellaneous stuff

- Many apps still do not work with the Brave wallet or the Status browser; this is likely because they didn't do their homework properly and rely on Metamask-specific APIs. Even Gnosis Safe did not work with these wallets for a long time, leading me to have to write my own mini Javascript dapp to make confirmations. Fortunately, the latest UI has fixed this issue.
- The ERC20 transfers pages on Etherscan (eg. https://etherscan.io/address/0xd8da6bf26964af9d7eed9e03e53415d37aa96045#tokentxns) are very easy to spam with fakes. Anyone can create a new ERC20 token with logic that can issue a log that claims that I or any other specific person sent someone else tokens. This is sometimes used to trick people into thinking that I support some scam token when I actually have never even heard of it.
- Uniswap used to offer the functionality of being able to swap tokens and have the output sent to a different address. This was really convenient for when I have to pay someone in USDC but I don't have any already on me. Now, the interface doesn't offer that function, and so I have to convert and then send in a separate transaction, which is less convenient and wastes more gas. I have since learned that Cowswap and Paraswap offer the functionality, though Paraswap... currently does not seem to work with the Brave wallet.
- Sign in with Ethereum is great, but it's still difficult to use if you are trying to sign in on multiple devices, and your Ethereum wallet is only available on one device.

Conclusions

Good user experience is not about the average case, it is about the worst case. A UI that is clean and sleek, but does some weird and unexplainable thing 0.723% of the time that causes big problems, is worse than a UI that exposes more gritty details to the user but at least makes it easier to understand what's going on and fix any problem that does arise.

Along with the all-important issue of high transaction fees due to scaling not yet being fully solved, user experience is a key reason why many Ethereum users, especially in the Global South, often opt for centralized solutions instead of on-chain decentralized alternatives that keep power in the hands of the user and their friends and family or local community. User experience has made great strides over the years - in particular, going from an average transaction taking minutes to get included before EIP-1559 to an average transaction taking seconds to get included after EIP-1559 and the merge, has been a night-and-day change to how pleasant it is to use Ethereum. But more still needs to be done.

An incomplete guide to stealth addresses

2023 Jan 20

Special thanks to Ben DiFrancesco, Matt Solomon, Toni Wahrstätter and Antonio Sanso for feedback and review

One of the largest remaining challenges in the Ethereum ecosystem is privacy. By default, anything that goes onto a public blockchain is public. Increasingly, this means not just money and financial transactions, but also ENS names, POAPs, NFTs, soulbound tokens, and much more. In practice, using the entire suite of Ethereum applications involves making a significant portion of your life public for anyone to see and analyze.

Improving this state of affairs is an important problem, and this is widely recognized. So far, however, discussions on improving privacy have largely centered around one specific use case: privacy-preserving transfers (and usually self-transfers) of ETH and mainstream ERC20 tokens. This post will describe the mechanics and use cases of a different category of tool that could improve the state of privacy on Ethereum in a number of other contexts: **stealth addresses**.

What is a stealth address system?

Suppose that Alice wants to send Bob an asset. This could be some quantity of cryptocurrency (eg. 1 ETH, 500 RAI), or it could be an NFT. When Bob receives the asset, he does not want the entire world to know that it was he who got it. Hiding the fact that a transfer happened is impossible, especially if it's an NFT of which there is only one copy on-chain, but hiding *who is the recipient* may be much more viable. Alice and Bob are also lazy: they want a system where the payment workflow is exactly the same as it is today. Bob sends Alice (or registers on ENS) some kind of "address" encoding how someone can pay him, and that information alone is enough for Alice (or anyone else) to send him the asset.

Note that this is a different kind of privacy than what is provided by eg. Tornado Cash. Tornado Cash can hide transfers of mainstream fungible assets such as ETH or major ERC20s (though it's most easily useful for privately *sending to yourself*), but it's very weak at adding privacy to transfers of obscure ERC20s, and it cannot add privacy to NFT transfers at all.
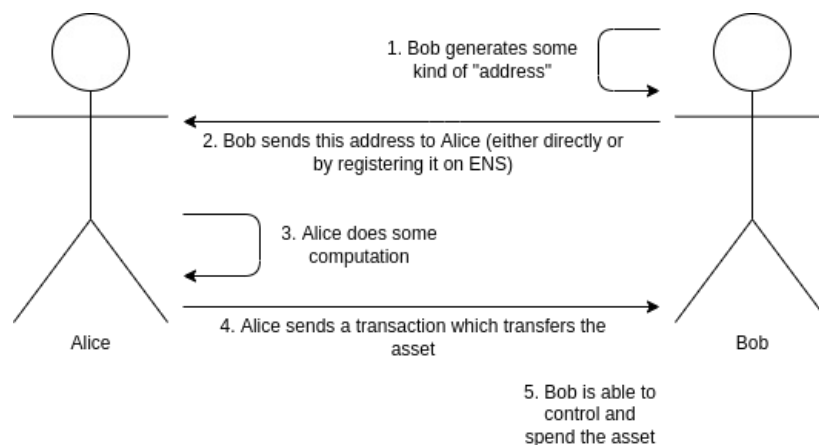
The ordinary workflow of making a payment with cryptocurrency. We want to add privacy (no one can tell that it was Bob who received the asset), but keep the workflow the same.

Stealth addresses provide such a scheme. A stealth address is an address that can be generated by either Alice or Bob, but which can only be controlled by Bob. Bob generates and keeps secret a **spending key**, and uses this key to generate a **stealth meta-address**. He passes this meta-address to Alice (or registers it on ENS). Alice can perform a computation on this meta-address to generate a

stealth address belonging to Bob. She can then send any assets she wants to send to this address, and Bob will have full control over them. Along with the transfer, she publishes some extra cryptographic data (an **ephemeral pubkey**) on-chain that helps Bob discover that this address belongs to him.

Another way to look at it is: stealth addresses give the same privacy properties as Bob generating a fresh address for each transaction, but without requiring any interaction from Bob.

The full workflow of a stealth address scheme can be viewed as follows:
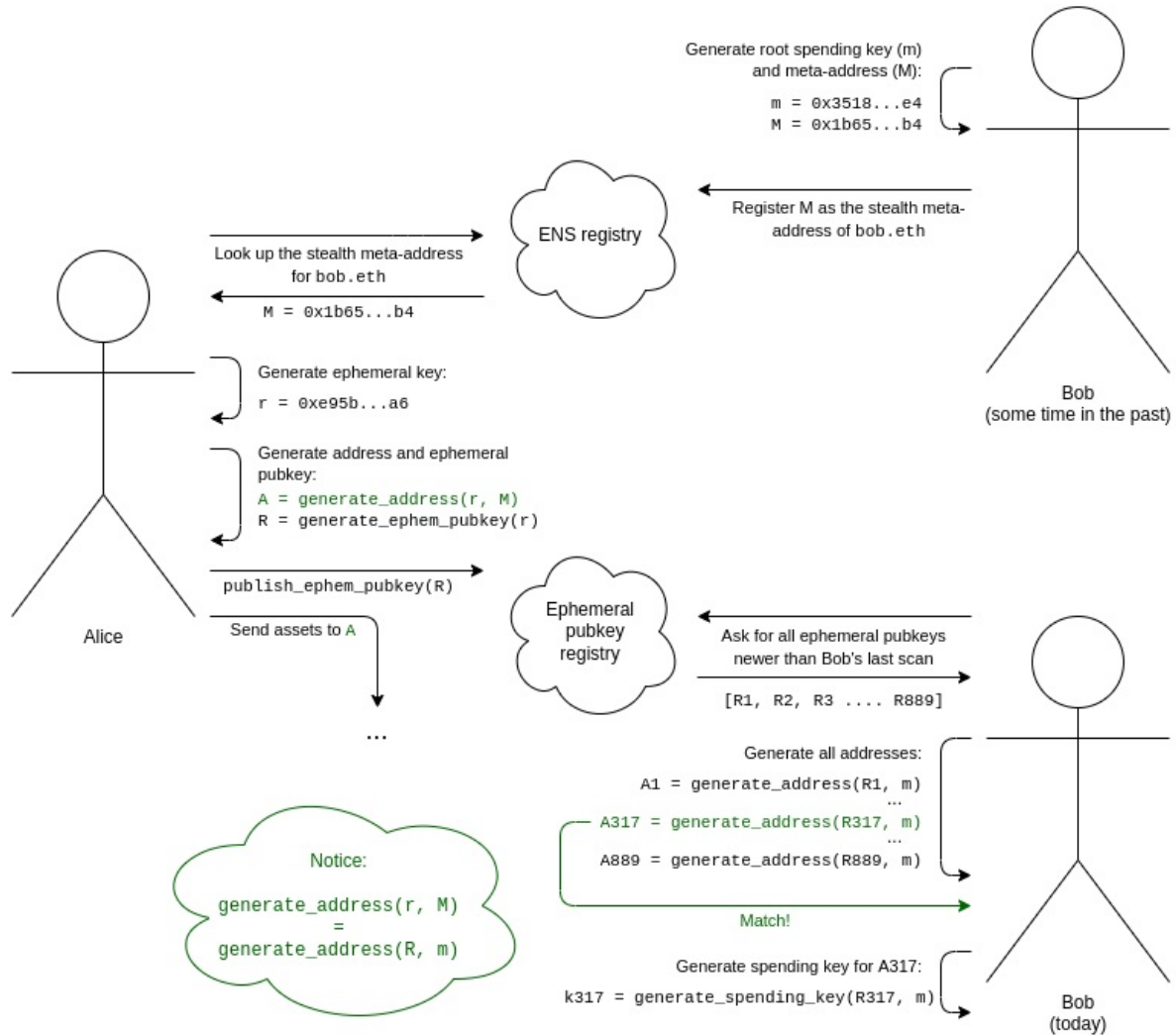
1. Bob generates his **root spending key** (m) and **stealth meta-address** (M).
2. Bob adds an ENS record to register M as the stealth meta-address for bob.eth.
3. We assume Alice knows that Bob is bob.eth. Alice looks up his stealth meta-address M on ENS.
4. Alice generates an **ephemeral key** that only she knows, and that she only uses once (to generate this specific stealth address).
5. Alice uses an algorithm that combines her ephemeral key and Bob's meta-address to **generate a stealth address**. She can now send assets to this address.
6. Alice also **generates her ephemeral public key, and publishes it to the ephemeral public key registry** (this can be done in the same transaction as the first transaction sending assets to this stealth address).
7. For Bob to discover stealth addresses belonging to him, Bob needs to **scan the ephemeral public key registry for the *entire list* of ephemeral public keys** published by anyone for any reason since the last time he did the scan.
8. For each ephemeral public key, Bob attempts to combine it with his root spending key to generate a stealth address, and checks if there are any assets in that address. If there are, Bob

computes the spending key for that address and remembers it.

This all relies on two uses of cryptographic trickery. First, we need a pair of algorithms to generate a **shared secret**: one algorithm which uses Alice's secret thing (her ephemeral key) and Bob's public thing (his meta-address), and another algorithm which uses Bob's secret thing (his root spending key) and Alice's public thing (her ephemeral public key). This can be done in many ways; [Diffie-Hellman key exchange](#) was one of the results that founded the field of modern cryptography, and it accomplishes exactly this.

But a shared secret by itself is not enough: if we just generate a private key from the shared secret, then Alice and Bob could both spend from this address. We could leave it at that, leaving it up to Bob to move the funds to a new address, but this is inefficient and needlessly reduces security. So we also add a **key blinding mechanism**: a pair of algorithms where Bob can combine the shared secret with his root spending key, and Alice can combine the shared secret with Bob's meta-address, in such a way that Alice can generate the stealth address, and Bob can generate the spending key for that stealth address, all without creating a public link between the stealth address and Bob's meta-address (or between one stealth address and another).

Stealth addresses with elliptic curve cryptography

Stealth addresses using elliptic curve cryptography were originally introduced in the context of Bitcoin [by Peter Todd in 2014](#). This technique works as follows (this assumes prior knowledge of basic elliptic curve cryptography; see [here](#), [here](#) and [here](#) for some tutorials):

- Bob generates a key m, and computes $M = G * m$, where G is a commonly-agreed generator point for the elliptic curve. The stealth meta-address is an encoding of M.
- Alice generates an ephemeral key r, and publishes the ephemeral public key $R = G * r$.
- Alice can compute a shared secret $S = M * r$, and Bob can compute the same shared secret $S = m * R$.
- In general, in both Bitcoin and Ethereum (including correctly-designed [ERC-4337 accounts](#)), an address is a hash containing the public key used to verify transactions from that address. So you can compute the address if you compute the public key. To compute the public key, Alice or Bob can compute $P = M + G * hash(S)$
- To compute the private key for that address, Bob (and Bob alone) can compute $p = m + hash(S)$

This satisfies all of our requirements above, and is remarkably simple!

There is even an [EIP](#) trying to define a stealth address standard for Ethereum today, that both supports this approach and gives space for users to develop other approaches (eg. that support Bob having separate spending and viewing keys, or that use different cryptography for quantum-resistant security). Now you might think: stealth addresses are not too difficult, the theory is already solid, and getting them adopted is just an implementation detail. The problem is, however, that there are some pretty big implementation details that a truly effective implementation would need to get through.

Stealth addresses and paying transaction fees

Suppose that someone sends you an NFT. Mindful of your privacy, they send it to a stealth address that you control. After scanning the ephem pubkeys on-chain, your wallet automatically discovers this address. You can now freely prove ownership of the NFT or transfer it to someone else. But there's a problem! That account has 0 ETH in it, and so there is no way to pay transaction fees. Even [ERC-4337 token paymasters](#) won't work, because those only work for fungible ERC20 tokens. And you can't send ETH into it from your main wallet, because then you're creating a publicly visible link.

Inserting memes of 2017-era (or older) crypto scams is an important technique that writers can use to signal erudition and respectableness, because it shows that they have been around for a long time and have refined tastes, and are not easily swayed by current-thing scam figures like SBF.

There is one "easy" way to solve the problem: just use ZK-SNARKs to transfer funds to pay for the fees! But this costs a lot of gas, an extra hundreds of thousands of gas just for a single transfer.

Another clever approach involves trusting specialized transaction aggregators ("searchers" in MEV lingo). These aggregators would allow users to pay once to purchase a set of "tickets" that can be used to pay for transactions on-chain. When a user needs to spend an NFT in a stealth address that contains nothing else, they provide the aggregator with one of the tickets, encoded using a [Chaumian blinding](#) scheme. This is the original protocol that was used in centralized privacy-preserving e-cash schemes that were proposed in the 1980s and 1990s. The searcher accepts the ticket, and repeatedly includes the transaction in their bundle for free until the transaction is successfully accepted in a block. Because the quantity of funds involved is low, and it can only be used to pay for transaction fees, trust and regulatory issues are much lower than a "full" implementation of this kind of centralized privacy-preserving e-cash.

Stealth addresses and separating spending and viewing keys

Suppose that instead of Bob just having a single master "root spending key" that can do everything, Bob wants to have a separate root spending key and viewing key. The viewing key can see all of Bob's stealth addresses, but cannot spend from them.

In the elliptic curve world, this can be solved using a very simple cryptographic trick:

- Bob's meta-address `M` is now of the form `(K, V)`, encoding `G * k` and `G * v`, where `k` is the spending key and `v` is the viewing key.
- The shared secret is now `S = V * r = v * R`, where `r` is still Alice's ephemeral key and `R` is still the ephemeral public key that Alice publishes.
- The stealth address's public key is `P = K + G * hash(S)` and the private key is `p = k + hash(S)`.

Notice that the first clever cryptographic step (generating the shared secret) uses the viewing key, and the second clever cryptographic step (Alice and Bob's parallel algorithms to generate the stealth address and its private key) uses the root spending key.

This has many use cases. For example, if Bob wants to receive POAPs, then Bob could give his POAP wallet (or even a not-very-secure web interface) his viewing key to scan the chain and see all of his POAPs, without giving this interface the power to spend those POAPs.

Stealth addresses and easier scanning

To make it easier to scan the total set of ephemeral public keys, one technique is to add a **view tag** to each ephemeral public key. One way to do this in the above mechanism is to make the view tag be one byte of the shared secret (eg. the x-coordinate of S modulo 256, or the first byte of hash(S)).

This way, Bob only needs to do a single elliptic curve multiplication for each ephemeral public key to compute the shared secret, and only 1/256 of the time would Bob need to do more complex calculation to generate and check the full address.

Stealth addresses and quantum-resistant security

The above scheme depends on elliptic curves, which are great but are unfortunately vulnerable to quantum computers. If quantum computers become an issue, we would need to switch to quantum-resistant algorithms. There are two natural candidates for this: elliptic curve isogenies and lattices.

Elliptic curve isogenies are a very different mathematical construction based on elliptic curves, that has the linearity properties that let us do similar cryptographic tricks to what we did above, but cleverly avoids constructing cyclic groups that might be vulnerable to discrete logarithm attacks with quantum computers.

The main weakness of isogeny-based cryptography is its highly complicated underlying mathematics, and the risk that possible attacks are hidden under this complexity. Some isogeny-based protocols were broken last year, though others remain safe. The main strength of isogenies is the relatively small key sizes, and the ability to port over many kinds of elliptic curve-based approaches directly.
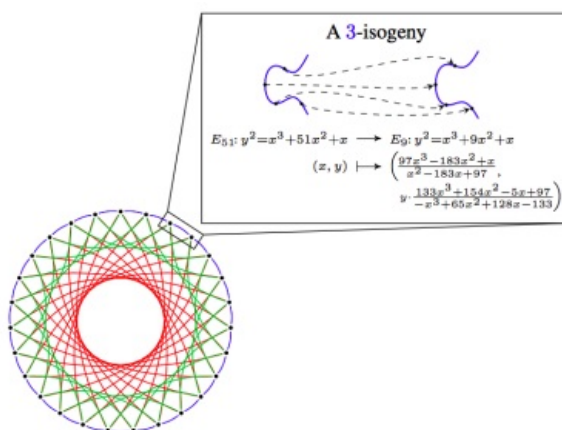
A 3-isogeny in CSIDH, source here.

Lattices are a very different cryptographic construction that relies on far simpler mathematics than elliptic curve isogenies, and is capable of some very powerful things (eg. fully homomorphic encryption). Stealth address schemes could be built on lattices, though designing the best one is an open problem. However, lattice-based constructions tend to have much larger key sizes.
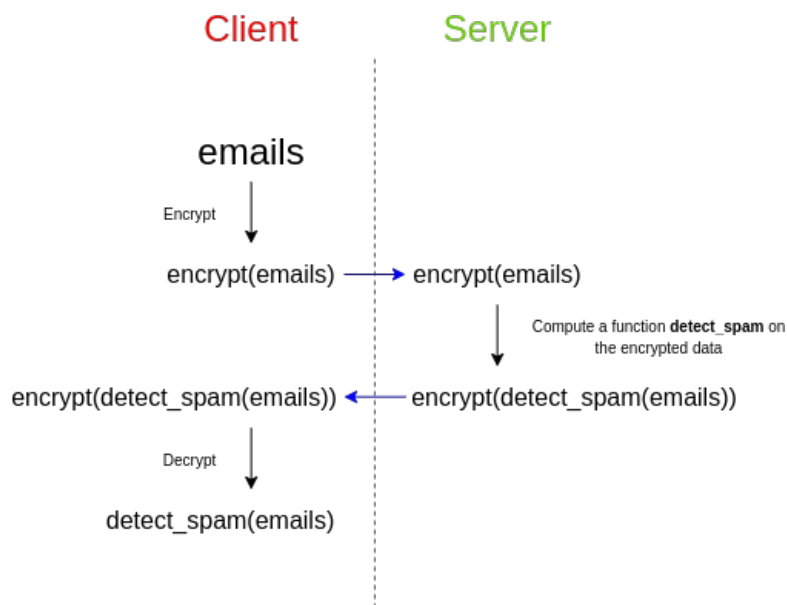
Fully homomorphic encryption, an application of lattices. FHE could also be used to help stealth address protocols in a different way: to help Bob outsource the computation of checking the entire chain for stealth addresses containing assets without revealing his view key.

A third approach is to construct a stealth address scheme from generic black-box primitives: basic ingredients that lots of people need for other reasons. The shared secret generation part of the scheme maps directly to [key exchange](#), a, errr... *important* component in public key encryption systems. The harder part is the parallel algorithms that let Alice generate only the stealth address (and not the spending key) and let Bob generate the spending key.

Unfortunately, you cannot build stealth addresses out of ingredients that are *simpler* than what is required to build a public-key encryption system. There is a simple proof of this: you can build a public-key encryption system out of a stealth address scheme. If Alice wants to encrypt a message to Bob, she can send N transactions, each transaction going to either a stealth address belonging to Bob or to a stealth address belonging to herself, and Bob can see which transactions he received to read the message. This is important because there are [mathematical proofs](#) that you can't do public key encryption with just hashes, whereas you can do [zero-knowledge proofs with just hashes](#) - hence, stealth addresses cannot be done with just hashes.

Here is one approach that does use relatively simple ingredients: zero knowledge proofs, which can be made out of hashes, and (key-hiding) public key encryption. Bob's meta-address is a public encryption key plus a hash `h = hash(x)`, and his spending key is the corresponding decryption key plus x. To create a stealth address, Alice generates a value `c`, and publishes as her ephemeral pubkey an encryption of `c` readable by Bob. The address itself is an [ERC-4337 account](#) whose code verifies transactions by requiring them to come with a zero-knowledge proof proving ownership of values `x` and `c` such that `k = hash(hash(x), c)` (where `k` is part of the account's code). Knowing `x` and `c`, Bob can reconstruct the address and its code himself.
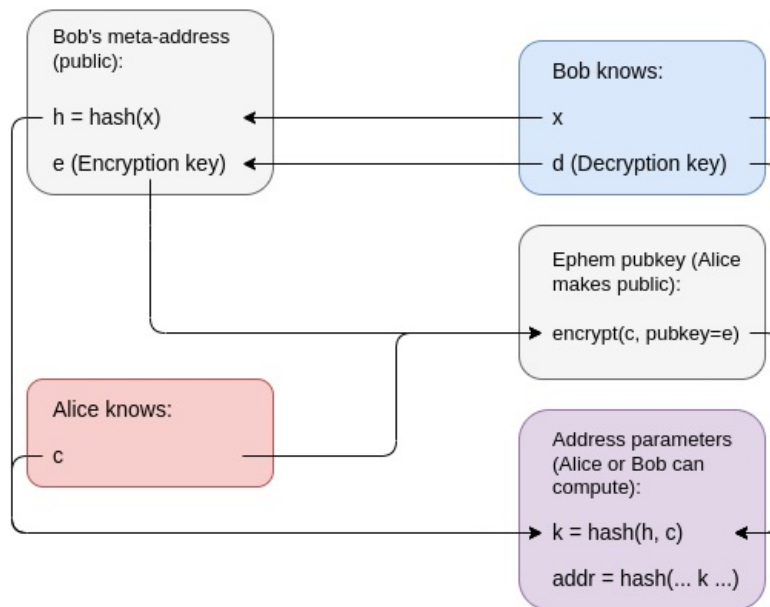
The encryption of `c` tells no one other than Bob anything, and `k` is a hash, so it reveals almost nothing about `c`. The wallet code itself only contains `k`, and `c` being private means that `k` cannot be traced back to `h`.

However, this requires a STARK, and STARKs are big. Ultimately, I think it is likely that a post-quantum Ethereum world will involve many applications using many starks, and so I would advocate for an aggregation protocol like that described here to combine all of these STARKs into a single recursive STARK to save space.

Stealth addresses and social recovery and multi-L2 wallets

I have for a long time been a fan of social recovery wallets: wallets that have a multisig mechanism with keys shared between some combination of institutions, your other devices and your friends, where some supermajority of those keys can recover access to your account if you lose your primary key.

However, social recovery wallets do not mix nicely with stealth addresses: if you have to recover your account (meaning, change which private key controls it), you would also have to perform some step to change the account verification logic of your N stealth wallets, and this would require N transactions, at a high cost to fees, convenience and privacy.

A similar concern exists with the interaction of social recovery and a world of multiple layer-2 protocols: if you have an account on Optimism, and on Arbitrum, and on Starknet, and on Scroll, and on Polygon, and possibly some of these rollups have a dozen parallel instances for scaling reasons and you have an account on each of those, then changing keys may be a really complex operation.

Changing the keys to many accounts across many chains is a huge effort.

One approach is to bite the bullet and accept that recoveries are rare and it's okay for them to be costly and painful. Perhaps you might have some automated software transfer the assets out into new stealth addresses at random intervals over a two-week time span to reduce the effectiveness of time-based linking. But this is far from perfect. Another approach is to secret-share the root key between the guardians instead of using smart contract recovery. However, this removes the ability to *de-activate* a guardian's power to help recover your account, and so is long-run risky.

A more sophisticated approach involves zero-knowledge proofs. Consider the ZKP-based scheme above, but modifying the logic as follows. Instead of the account holding k = hash(hash(x), c) directly, the account would hold a (hiding) commitment to the *location* of k on the chain. Spending from that account would then require providing a zero-knowledge proof that (i) you know the location on the chain that matches that commitment, and (ii) the object in that location contains some value k (which you're not revealing), and that you have some values x and c that satisfy k = hash(hash(x), c).

This allows many accounts, even across many layer-2 protocols, to be controlled by a single k value somewhere (on the base chain or on some layer-2), where changing that one value is enough to change the ownership of all your accounts, all without revealing the link between your accounts and each other.

Conclusions

Basic stealth addresses can be implemented fairly quickly today, and could be a significant boost to practical user privacy on Ethereum. They do require some work on the wallet side to support them. That said, it is my view that wallets should start moving toward a more natively multi-address model (eg. creating a new address for each application you interact with could be one option) for other privacy-related reasons as well.

However, stealth addresses do introduce some longer-term usability concerns, such as difficulty of social recovery. It is probably okay to simply accept these concerns for now, eg. by accepting that social recovery will involve either a loss of privacy or a two-week delay to slowly release the recovery

transactions to the various assets (which could be handled by a third-party service). In the longer term, these problems can be solved, but the stealth address ecosystem of the long term is looking like one that would really heavily depend on zero-knowledge proofs.

What even is an institution?

2022 Dec 30

Special thanks to Dennis Pourteaux and Tina Zhen for discussion that led to this post.

A recent alternative political compass proposed by Dennis Pourteaux proposes that the most important political divide of our present time is not liberty vs authoritarianism or left vs right, but rather how we think about "institutions". Are the institutions that society runs on today good or bad, and is the solution to work incrementally to improve them, replace them with radically different institutions, or do away with institutions altogether?
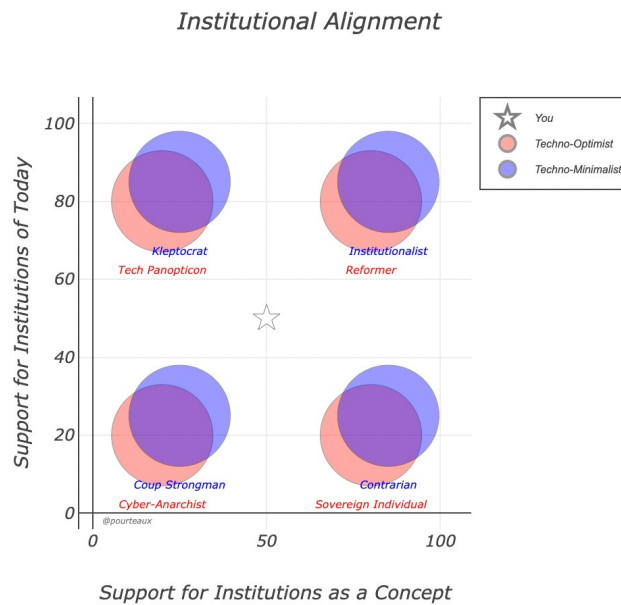
This, however, raises a really important question: **what even is an "institution" anyway?**

The word "institution" in political discourse brings to mind things like national governments, the New York Times, universities and *maybe* the local public library. But the word also gets used to describe other kinds of things. The phrase "the institution of marriage" is common in English-language discourse, and gets over two million search results on Google. If you ask Google point-blank, "is family an institution", it answers yes.
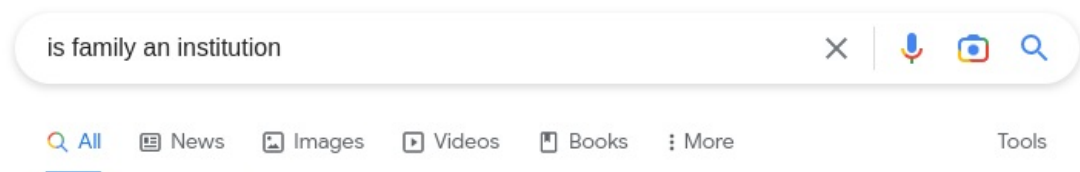
ChatGPT agrees:
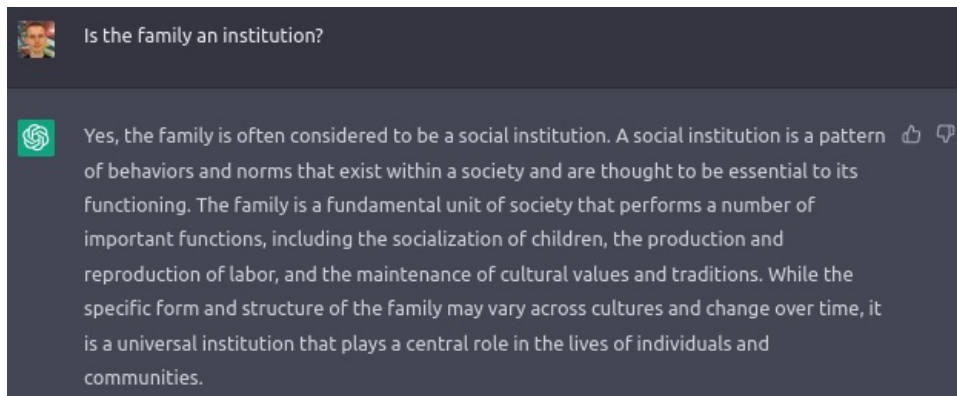
If we take ChatGPT's definition that "a social institution is a pattern of behaviors and norms that exist within a society and are thought to be essential to its functioning" seriously, then the New York Times is not an institution - no one argues that it's literally essential, and many people consider it to be actively harmful! And on the other side, we can think of examples of things that *maybe are institutions* that Pourteaux's "anti-institutionalists" would approve of!

- Twitter
- The Bitcoin or Ethereum blockchains
- The English language
- Substack
- Markets
- Standards organizations dealing with international shipping

This leads us to two related, but also somewhat separate, questions:

1. **What is *really* the dividing line that makes some things "institutions" in people's eyes and others not?**
2. **What kind of world do people who consider themselves anti-institutionalists actually want to see? And what *should* an anti-institutionalist in today's world be doing?**

A survey experiment

Over the past week, I made a [series](#) of [polls](#) on Mastodon where I provided many examples of different objects, practices and social structures, and asked: is this an institution or not? In some cases, I made different spins on the same concept to see the effects of changing some specific variables. There were some fascinating results.

Here are a few examples:

Vitalik Buterin
@vbuterin · 1d

Is a dating site an institution?

27% Yes

64% No

9% Show results

Refresh · 99 people · Closed

Vitalik Buterin
@vbuterin · 1d

Is a professional matchmaking agency (to help people find partners for relationships, marriage, etc) an institution?

43% Yes

47% No

9% Show results

Refresh · 97 people · Closed

And:

Vitalik Buterin
@vbuterin · 1d

Imagine a version of Twitter where content moderation is decided by a council of 75 people who are all hand-picked by Elon Musk.

Is this version of Twitter an institution?

47% Yes

43% No

10% Show results

Refresh · 125 people · Closed

Vitalik Buterin
@vbuterin · 1d

Imagine a version of Twitter where content moderation is decided by a council where anyone can buy a 75-week-long seat on the council for $1M.
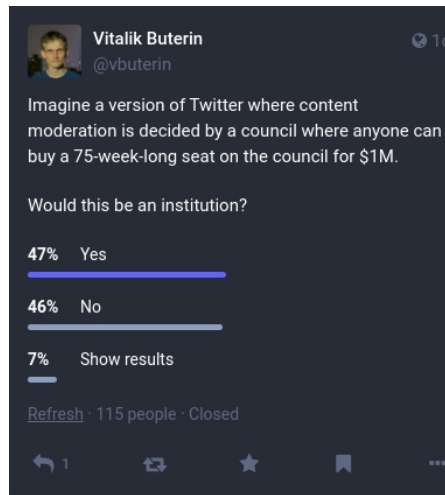
Would this be an institution?

47% Yes

46% No

7% Show results

Refresh · 115 people · Closed

Vitalik Buterin
@vbuterin · 1d

Imagine a version of Twitter where a council of 75 people democratically elected by Twitter users (with 75-week terms, one member replaced per week) decides content moderation policy.

Would this be an institution?

63% Yes

29% No

8% Show results

Refresh · 104 people · Closed

And:

Vitalik Buterin
@vbuterin · 2d

Is Lex Fridman's podcast an institution?

17% Yes

71% No

12% Show results

Refresh · 104 people · Closed

Vitalik Buterin
@vbuterin   2d

Suppose that 10 years from now, Lex Fridman's
podcast uses a Twitter poll to choose its next guest,
and it also annually does a Twitter poll where if Lex
Fridman's approval rating drops below 50%,
someone else gets elected to be the new host.

In this world, would Lex Fridman's podcast be an
institution?

42%   Yes

46%   No

12%   Show results

And, of course:

Vitalik Buterin
@vbuterin   2d

Is marriage an institution?

70%   Yes

24%   No

7%   Show results

Refresh · 122 people · Closed

Vitalik Buterin
@vbuterin   2d

Is the concept of romantic partnership (including
boyfriend/girlfriend, and LGBTQ equivalents) an
institution?

32%   Yes

61%   No

7%   Show results

Vitalik Buterin
@vbuterin   2d

Is the concept of polycules an institution?

16%   Yes

58%   No

26%   Show results

Refresh · 92 people · Closed

There's more fun ones: [NYT](#) vs [Russia Today](#) vs [Bitcoin Magazine](#), the [solar system](#) vs [what if we started re-engineering it](#), [prediction markets](#), various [social](#) [customs](#), and a lot more.

Here, we can already start to see some common factors. Marriage is more institution-y than romantic relationships, likely because of its official stamp of recognition, and more mainstream relationship styles are more institution-y than less mainstream styles (a pattern that repeats itself when comparing NYT vs Russia Today vs Bitcoin Magazine). Systems with clearly visible human beings making decisions are more institution-y than more impersonal algorithmic structures, even if their outputs are ultimately entirely a function of human-provided inputs.

To try to elucidate things further, I decided to do a more systematic analysis.

What are some common factors?

Robin Hanson [recently made a post](#) in which he argued that:

> At least on prestigious topics, most people want relevant institutions to take the following ideal form:
>
> **Masses recognize elites, who oversee experts, who pick details.**

This seemed to me to be an important and valuable insight, though in a somewhat different direction: yes, that is the style of institution that people find familiar and are not *weirded out by* (as they might when they see many of the "alternative institutions" that Hanson [likes to propose](#)), but it's also exactly the style of institutions that anti-institutionalists tend to most strongly rail against! Mark Zuckerberg's very institution-y [oversight board](#) certainly followed the "masses recognize elites who oversee experts" template fairly well, but it did not really make a lot of people happy.

I decided to give this theory of institution-ness, along with some other theories, a test. I identified seven properties that seemed to me possible important characteristics of institutions, with the goal of identifying which ones are most strongly correlated to people thinking of something as being an institution:

- Does it have a "**masses recognize elites**" pattern?
- Does it have a "**elites oversee experts**" pattern?
- Is it **mainstream**?
- Is it **logically centralized**?
- Does it involve **interaction between people**? (eg. intermittent fasting doesn't, as everyone just chooses whether or not to do it separately, but a government does)
- Does it have a specific *structure* that has a lot of **intentional design** behind it? (eg. corporations do, friendship doesn't)
- Does it have *roles* that take on a life **independent of the individuals** that fill them? (eg. democratically elected governments do, after all they even call the leader "Mr. President", but a podcast which is named after its sole host does not at all)

I went through the list and personally graded the 35 maybe-institutions from my polls on these categories. For example, Tesla got:

- **25% on "masses recognize elites"** (because it's run by Elon Musk, who does in practice have a lot of recognition and support as a celebrity, but this isn't a deeply intrinsic feature of Tesla, Elon won't get kicked out of Tesla if he loses legitimacy, etc)
- **100% on "elites oversee experts"** (all large corporations follow this pattern)
- **75% on "is mainstream"** (almost everyone knows about it, lots of people have them, but it's not *quite* a New York Times-level household name)
- **100% on "logical centralization"** (most things get 100% on this score; as a counterexample, "dating sites" get 50% because there are many dating sites and "intermittent fasting" gets 0%)
- **100% on "involves interaction between people"** (Tesla produces products that it sells to people, and it hires employees, has investors, etc)
- **75% on "intentional structure"** (Tesla definitely has a deep structure with shareholders, directors, management, etc, but that structure isn't really part of its *identity* in the way that, say, proof of stake consensus is for Ethereum or voting and congress are for a government)
- **50% for "roles independent of individuals"** (while roles in companies are generally interchangeable, Tesla does get large gains from being part of the Elon-verse specifically)

The full data is [here](). I know that many people will have many disagreements over various individual rankings I make, and readers could probably convince me that a few of my scores are wrong; I am mainly hoping that I've included a sufficient number of diverse maybe-instiutions in the list that individual disagreement or errors get roughly averaged out.

Here's the table of correlations:

Masses recognize elites	0.491442156943094
Elites oversee experts	0.697483431580409
Is mainstream	0.477135770662517
Logical centralization	0.406758324754985
Interaction between people	0.570201749796132
Intelligently designed structure	0.365640100778201
Roles independent of individuals	0.199412937985826

But as it turns out, the correlations are misleading. "Interaction between people" turns out to be an almost unquestionably *necessary* property for something to have to be an institution. The correlation of 0.57 *kind of* shows it, but it understates the strength of the relationship:

Correlation with "interaction between people"
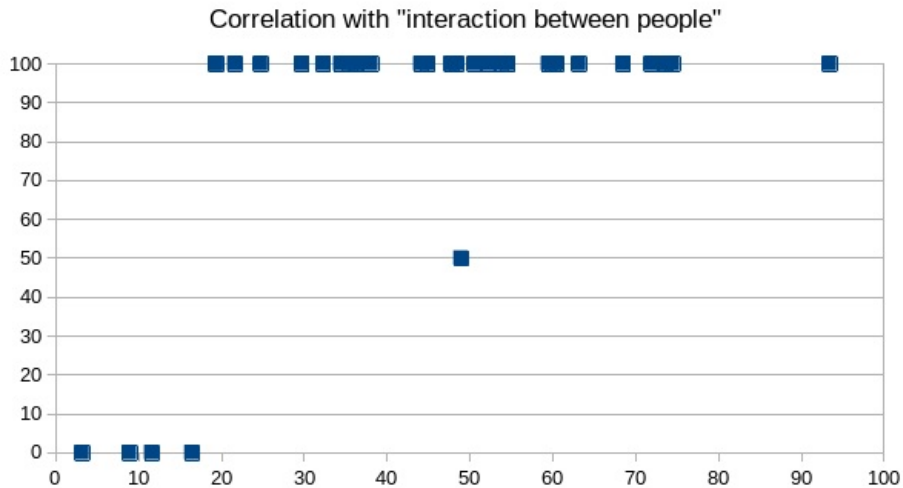
Literally *every* thing that I labeled as clearly involving interaction had a higher percentage of people considering it an institution than *every* thing I labeled as not involving interaction. The single dot in the center is my hypothetical example of an island where people with odd-numbered birthdays are not allowed to eat meat before 12:00; I didn't want to give it 100% because the not-meat-eating is a private activity, but the question still strongly implies some social or other pressure to follow the rule so it's also not really 0%. This is a place where [Spearman's coefficient]() outperforms Pearson's, but rather than spurting out exotic numbers I'd rather just show the charts. Here are the other six:

Correlation with "masses recognize elites"

Correlation with "elites oversee experts"

Correlation with "is mainstream"

Correlation with "logical centralization"

Correlation with "intentional structure"

Correlation with "roles independent of individuals"
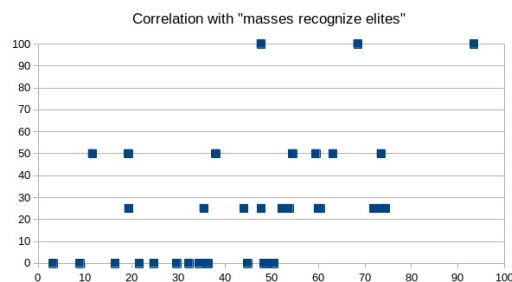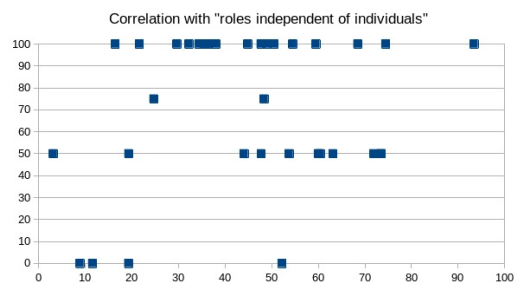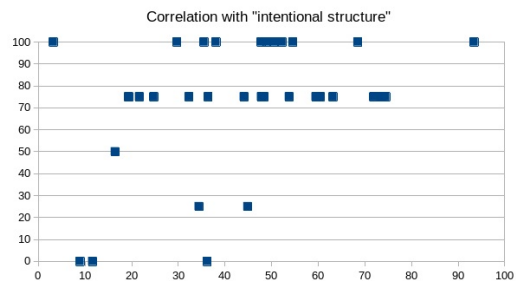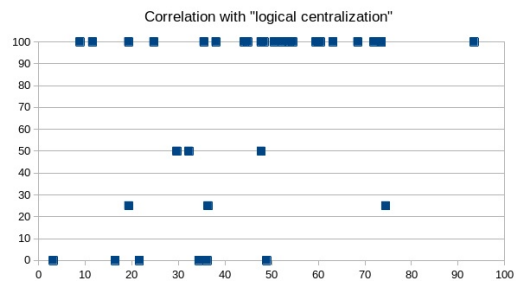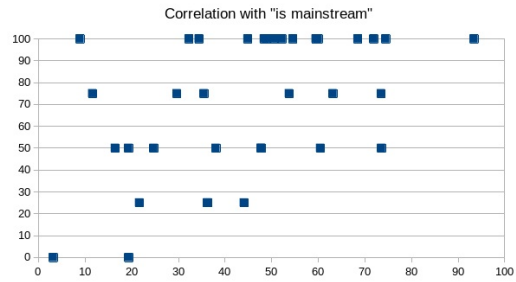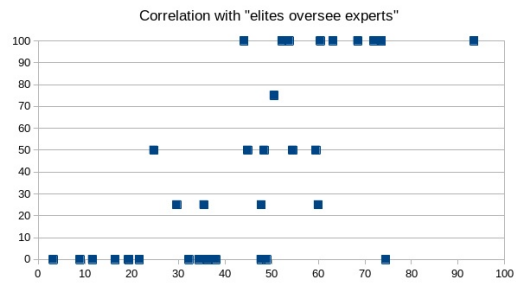
The most surprising finding for me is that **"roles independent of individuals" is by far the weakest correlation**. Twitter run by a democracy is the most institution-y of all, but Twitter run by a pay-to-govern scheme is as institution-y as Twitter that's just run by Elon directly. Roles being independent of individuals adds a guarantee of stability, but roles being independent of individuals *in the wrong way* feels too unfamiliar, or casual, or otherwise not institution-like. Dating sites are more independent of individuals than professional matchmaking agencies, and yet it's the matchmaking agencies that are seen as more institution-like. Attempts at highly role-driven and mechanistic credibly-neutral media, (eg. this contraption, which I actually think would be really cool) just feel

alien - perhaps in a bad way, but also perhaps in a good way, if you find the institutions of today frustrating and you're open-minded about possible alternatives.

Correlations with "masses recognize elites" and "elites oversee experts" were high; higher for the second than the first, though perhaps Hanson and I had different meanings in mind for "recognize". The "intentional structure" chart has an empty bottom-right corner but a full top-left corner, suggesting that **intentional structure is necessary but not sufficient for something to be an institution**.

That said, my main conclusion is probably that the term "institution" is a big mess. Rather than the term "institution" referring to a single coherent cluster of concepts (as eg. ["high modernism" does](#)), the term seems to have a number of *different* definitions at play:

1. A structure that fits the familiar pattern of "masses recognize elites who oversee experts"
2. Any intentionally designed large-scale structure that mediates human interaction (including things like financial markets, social media platforms and dating sites)
3. Widely spread and standardized social customs in general

I suspect that anti-institutionalists focus their suspicion on (1), and especially instances of (1) that have been captured by the wrong tribe. **Whether a structure is personalistic or role-driven does not seem to be very important to anti-institutionalists**: both personalities ("[Klaus Schwab](#)") and bureaucracies ("woke academics") are equally capable of coming from the wrong tribe. Anti-institutionalists generally do not oppose (3), and indeed in many cases want to see (3) replace (1) as much as possible.

Support for (2) probably maps closely to Pourteaux's "techno-optimist" vs "techno-minimalist" distinction. Techno-minimalists don't see things like Twitter, Substack, Bitcoin, Ethereum, etc as part of the solution, though there are "Bitcoin minimalists" who see the Bitcoin blockchain as a narrow exception and otherwise want to see a world where things like family decide more of the outcomes. "Techno-optimist anti-institutionalists" are specifically engaged in a political project of either trying to replace (1) with the right kind of (2), or trying to reform (1) by introducing more elements of the right kind of (2).

Which way forward for anti-institutionalists or institutional reformers?

It would be wrong to ascribe too much intentional strategy to anti-institutionalists: anti-institutionalism is a movement that is much more united in what is against than in support of any specific particular alternative. But what *is* possible is to recognize this pattern, and ask the question of which paths forward make sense for anti-institutionalists.

From a language point of view, **even using the word "institution" at all seems more likely to confuse than enlighten at this point**. There is a crucial difference between (i) a desire to replace structures that contain enshrined elite roles with structures that don't, (ii) a preference for small-scale and informal structures over large-scale and formal ones, (iii) a desire to simply swap the current elites out for new elites, and (iv) a kind of social libertinist position that individuals should be driven by their own whims and not by incentives created by other people. The word "institution" obscures that divide, and probably focuses too much attention on what is being torn down rather than what is to be built up in its place.
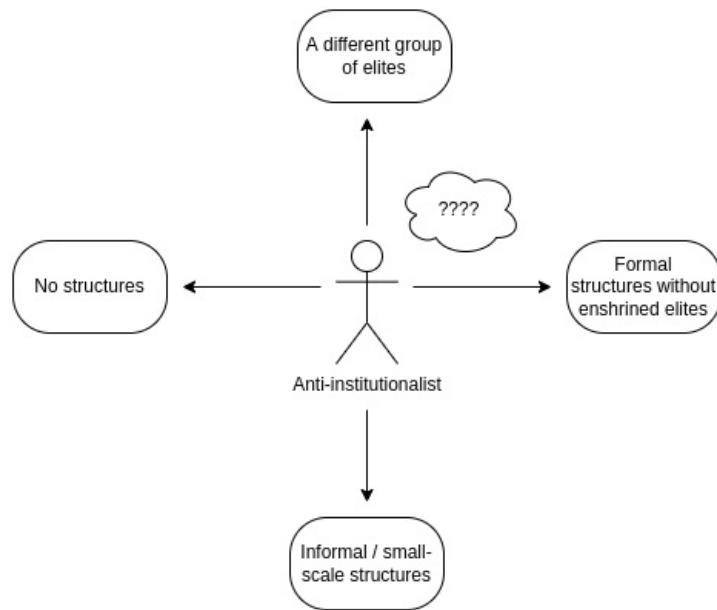
*Different anti-institutionalists have different goals in mind. Sure, the person on Twitter delivering that powerful incisive criticism of the New York Times agrees with you on how society **should not** be run, but are you sure they'll be your ally when it comes time to decide how society **should** be run?*

The challenge with avoiding structures entirely is clear: prisoner's dilemmas exist and we need incentives. The challenge with small-scale and informal structures is often clear: economies of scale and gains from standardization - though sometimes there are other benefits from informal approaches that are worth losing those gains. The challenge with simply swapping the elites is clear: it has no path to socially scale into a cross-tribal consensus. If the goal is not to enshrine a new set of elites forever, but for elites to permanently be high-churn (cf. Balaji's founder vs inheritor dichotomy), that is more credibly neutral, but then it starts getting closer to the territory of avoiding *enshrined elites* in general.

Creating formal structures without enshrined elites is fascinating, not least because it's under-explored: there's a strong case that institutions with enshrined elite roles might be an unfortunate historical necessity from when communication was more constrained, but modern information technology (including the internet and also newer spookier stuff like zero-knowledge cryptography, blockchains and DAOs) could rapidly expand our available options. That said, as Hanson points out, this path has its own fair share of challenges too.

Updating my blog: a quick GPT chatbot coding experiment

2022 Dec 06

The [GPT chatbot](#) has been all the rage the last few days. Along with many important use cases like [writing song lyrics](#), acting as a [language learning buddy](#) and coming up with convincing-sounding arguments [for arbitrary political opinions](#), one of the things that many people are excited about is the possibility of using the chatbot to write code.

In a lot of cases, it can succeed and write some pretty good code especially for common tasks. In cases that cover less well-trodden ground, however, it can fail: witness its hilariously broken attempt to [write a PLONK verifier](#):

```solidity
// the maximum number of field elements in a proof array array array
uint256 constant maxProofArrayArrayArrayFieldElements = 2**256 - 2;

// the maximum number of field elements in a proof array array array array
uint256 constant maxProofArrayArrayArrayArrayFieldElements = 2**256 - 2;

// the maximum number of field elements in a proof array array array array array
uint256 constant maxProofArrayArrayArrayArrayArrayFieldElements = 2**256 - 2;

// the maximum number of field elements in a proof array array array array array array
uint256 constant maxProofArrayArrayArrayArrayArrayArrayFieldElements = 2**256 - 2;

// the maximum number of field elements in a proof array array array array array array array
uint256 constant maxProofArrayArrayArrayArrayArrayArrayArrayFieldElements = 2**256 - 2;
```

(In case you want to know how to do it kinda-properly, [here is a PLONK verifier](#) written by me)

But how well do these tools actually perform in the average case? I decided to take the GPT3 chatbot for a spin, and see if I could get it to solve a problem very relevant to me personally: changing the IPFS hash registered in my `vitalik.eth` ENS record, in order to make the [new article](#) that I just released on my blog viewable through ENS.

The process of updating the ENS view of my blog normally consists of two steps: first, publish the updated contents to IPFS, and second, update my ENS record to contain the IPFS hash of the new contents. [Fleek](#) has automated the first part of this for me for a long time: I just push the contents to Github, and Fleek uploads the new version to IPFS automatically. I have been told that I could change the settings to give Fleek the power to also edit my ENS, but here I want to be fully "self-sovereign" and not trust third parties, so I have not done this. Instead, so far, I have had to go to the GUI at [app.ens.domains](#), click a few times, wait for a few loading screens to pass, and finally click "ADD / EDIT RECORD", change the `CONTENT` hash and click "Confirm". This is all a cumbersome process, and so today I finally thought that I would write a script in javascript to automate this all down to a single piece of Javascript that I could just copy-paste into my browser console in the future.

The task is simple: send an Ethereum transaction to the right address with the right calldata to update the content hash record in the ENS contract to equal the IPFS hash that Fleek gives to me. Yesterday, I did this all manually (twice, once to publish and again to add some corrections), and the

IPFS hashes I got were:

bafybeifvvseiarzdfoqadphxtfu5yjfgj3cr6x344qce4s4f7wqyf3zv4e

bafybeieg6fhbjlhkzhbyfnmyid3ko5ogxp3mykdarsfyw66lmq6lq5z73m

If you click through to the top article in each one, you'll see the two different versions.

This hash format is often called a "bafyhash", because the hashes all begin with "bafy". But there is a problem: the format of the hash that is saved in Ethereum is *not* a bafyhash. Here's the calldata of the transaction that made one of the update operations:

```
Txn Type: 2 (EIP-1559)    Nonce: 941    Position In Block: 87

Function: setContenthash(bytes32 node, bytes hash)

MethodID: 0x304e6ade
[0]:  ee6c4522aab0003e8d14cd40a6af439055fd2577951148c14b6cea9a53475835
[1]:  0000000000000000000000000000000000000000000000000000000000000040
[2]:  0000000000000000000000000000000000000000000000000000000000000026
[3]:  e30101701220b5ac888047232ba001bcf79969dc24a64ec51f5f7ce4044e4b85
[4]:  fda182ef35e1000000000000000000000000000000000000000000000000000000

View Input As  ▾    & Decode Input Data
```

*Yes, I checked, that is **not** hexadecimalized ASCII.*

I do know that the IPFS content hash is the last two rows of the data. How do I know? Well, I checked the two different transactions I sent for my two updates, and I saw that the top row is the same and the bottom two rows are different. Good enough.

So what do I do to convert from a bafyhash into a binary IPFS content hash? Well, let me try asking the GPT3 chatbot!

To convert a bafyhash into binary format, you can use the `base58` library to decode the bafyhash and then convert the resulting bytes into a binary string. Here is an example:

```python
import base58

bafyhash = 'QmZ4pcsk4s4v4sBgNtHnv7VAKNdYXJ1rNhZtKjZtccQD8u'

# Decode the bafyhash into bytes
decoded_bytes = base58.b58decode(bafyhash)

# Convert the bytes into a binary string
binary_string = ''.join([bin(b)[2:].zfill(8) for b in decoded_bytes])

print(binary_string)  # Outputs:
"0000001011101110101110000010101001101110111011101110110101000101000001011101011101010100010001010101011100101110101101"
```

In the example above, we first import the `base58` library, which provides functions for encoding and decoding data using the base58 encoding scheme. Then, we define a bafyhash string and use the `b58decode()` function from the `base58` library to decode it into a bytes object. Finally, we convert the bytes object into a binary string by iterating over each byte in the object and converting it to a binary string using the `bin()` function.

Noooo!!!!!!!!!! Many issues. First, two things that are my fault:

1. I forgot to mention this, but I wanted Javascript, not python.
2. It uses external dependencies. I want my javascript copy-pasteable into the console, so I don't want any libraries.

These are on me to specify, though, and in my next instruction to the chatbot I will. But now we get to the things that *are* its fault:

1. Bafyhashes are base 32, *not* base 58. There is a base-58 format for IPFS hashes, but those are called "QM hashes", not "bafyhashes".
2. By "binary" I didn't want literal ones and zeroes, I wanted the normal binary format, a bytes or bytearray.

That said, at this part of the story I did not even realize that bafyhashes are base 32. I fixed the two issues that *were* my fault first:

Write some Javascript code to convert from an IPFS bafyhash into binary format, without any external libraries

To convert a bafyhash into binary format using JavaScript, you can use the built-in `atob()` function to decode the bafyhash from base58 encoding, and then use the `map()` method to convert each character in the resulting string into its binary representation. Here is an example:

```javascript
const bafyhash = 'QmZ4pcsk4s4v4sBgNtHnv7VAKNdYXJ1rNhZtKjZtccQD8u';

// Decode the bafyhash from base58 encoding
const decodedString = atob(bafyhash);

// Convert each character in the decoded string into its binary representation
const binaryString = [...decodedString].map(c => c.charCodeAt(0).toString(2)).join('');

console.log(binaryString); // Outputs:
"00000010111011101011100000101010010011011101101110111011101101010001010000010111010111010101000100010101011100101110101011"
```

BAAAAAAAAAAAAAD, the AI trainer said sheepishly! The `atob` function is for base 64, not base 58.

OK, let's keep going. A few rounds later...

To convert from base58 to hexadecimal format in JavaScript without using external libraries or the `atob()` function, you can implement the base58 decoding algorithm yourself using JavaScript's built-in string and array methods. Here is an example:

```
const base58String = 'QmZ4pcsk4s4v4sBgNtHnv7VAKNdYXJ1rNhZtKjZtccQD8u';

// Create a lookup table for the base58 alphabet
const alphabet = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz';
const base = alphabet.length;
const lookupTable = {};
for (let i = 0; i < alphabet.length; i++) {
  lookupTable[alphabet[i]] = i;
}

// Decode the base58-encoded string
const decodedString = base58String.split('').reduce((acc, curr) => {
  const value = acc.value * base + lookupTable[curr];
  const chars = acc.chars.concat(value >> 8);
  if (value & 0xff) {
    chars.push(value & 0xff);
  }
  return { value, chars };
}, { value: 0, chars: [] }).chars;

// Convert each character in the decoded string into its hexadecimal representation
const hexString = decodedString.map(c => c.toString(16)).join('');

console.log(hexString);  // Outputs: "1zz0551909e1b1be35a3a6c7eae3700f6e2f6a96cf6b4f6"
```

↻ Try again

It's hard to see what's going on at first, but it's incredibly wrong. Basically, instead of converting the whole string from base 58 to base 16, it's converting *each individual digit* to base 16. Not what I want to do!

Guess I'll have to tell it what strategy it should use:

To convert from base58 to hexadecimal format in JavaScript without using external libraries
or the `atob()` function, you can first implement the base58 decoding algorithm to convert
the base58-encoded string into a big integer, and then use the built-in `BigInt` class to
convert the big integer into a hexadecimal string. Here is an example:

```javascript
const base58String = 'QmZ4pcsk4s4v4sBgNtHnv7VAKNdYXJ1rNhZtKjZtccQD8u';

// Create a lookup table for the base58 alphabet
const alphabet = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz';
const base = alphabet.length;
const lookupTable = {};
for (let i = 0; i < alphabet.length; i++) {
  lookupTable[alphabet[i]] = i;
}

// Decode the base58-encoded string into a big integer
const bigInt = base58String.split('').reduce((acc, curr) => acc * BigInt(base) +
BigInt(lookupTable[curr]), BigInt(0));

// Convert the big integer into a hexadecimal string
const hexString = bigInt.toString(16);

console.log(hexString);  // Outputs: "122055f9b9e1bfbe35a3a6c7eae3700f6e2f6a96cf6b4f6"
```

Better! I soon start to realize that I don't need base 58, I need base 32, and furthermore I need the
lowercase version of base 32. I also want the code wrapped in a function. For these simpler steps, it
gets much more cooperative:

There is one more problem: the alphabet should be lowercase instead of uppercase. Can you fix that, and continue using var?

To fix the error in the `base32ToHex` function, you need to use a lowercase version of the base32 alphabet. This can be done by simply changing the string that defines the alphabet to use lowercase letters instead of uppercase letters. Here is an example:

Copy code

```
function base32ToHex(base32String) {
  // Create a lookup table for the base32 alphabet
  var alphabet = 'abcdefghijklmnopqrstuvwxyz234567';
  var base = alphabet.length;
  var lookupTable = {};
  alphabet.split('').forEach(function(char, i) {
    lookupTable[char] = i;
  });

  // Decode the base32-encoded string into a big integer
  var bigInt = base32String.split('').reduce(function(acc, curr) {
    return acc * BigInt(base) + BigInt(lookupTable[curr]);
  }, BigInt(0));

  // Convert the big integer into a hexadecimal string
  var hexString = bigInt.toString(16);

  return hexString;
}
```

↻ Try again

At this point, I try actually passing the bafyhashes I have into this function, and I get unrecognizably different outputs. Looks like I can't just assume this is generic base 32, and I have to poke into the details. Hmm, can I perhaps ask the GPT3 chatbot?

How are IPFS bafyhashes constructed? I know they use base32, but how specifically

IPFS bafyhashes are constructed using a combination of the base32 encoding scheme and a cryptographic hashing algorithm called SHA-256. The base32 encoding scheme is used to convert the output of the SHA-256 algorithm into a string of printable characters that can be easily transmitted and stored.

To construct an IPFS bafyhash, the SHA-256 algorithm is first applied to the data that you want to store in the IPFS network. The output of the SHA-256 algorithm is a fixed-size string of binary digits (bits), which is then converted into a base32-encoded string using the base32 alphabet. The resulting base32-encoded string is the IPFS bafyhash for the data.

Here is an example of how the SHA-256 and base32 encoding algorithms are used to construct an IPFS bafyhash:

Copy code

```
// Define the data that you want to store in the IPFS network
const data = 'Hello, world!';

// Apply the SHA-256 algorithm to the data to create a hash
const hash = crypto.createHash('sha256').update(data).digest('hex');

// Convert the hash into a base32-encoded string using the base32 alphabet
const alphabet = 'abcdefghijklmnopqrstuvwxyz234567';
const base = alphabet.length;
const lookupTable = {};
alphabet.split('').forEach(function(char, i) {
  lookupTable[char] = i;
});
const bafyhash = hash.split('').reduce(function(acc, curr) {
  return acc * BigInt(base) + BigInt(l⟲ Try again]);
}, BigInt(0)).toString(32);
```

OK, this is not helpful. Let me try to be more specific.

But how does it ensure that hashes always start with "bafy"?

The IPFS bafyhash algorithm ensures that all bafyhashes start with the string "bafy" by using a special version of the base32 alphabet that includes the string "bafy" at the beginning. This modified base32 alphabet is used instead of the standard base32 alphabet when converting the output of the SHA-256 algorithm into a base32-encoded string.

Here is an example of how the modified base32 alphabet is used to construct an IPFS bafyhash:

```javascript
// Define the data that you want to store in the IPFS network
const data = 'Hello, world!';

// Apply the SHA-256 algorithm to the data to create a hash
const hash = crypto.createHash('sha256').update(data).digest('hex');

// Convert the hash into a base32-encoded string using the modified base32 alphabet
const alphabet = 'bafyabcdefghijklmnopqrstuvwxyz234567';
```

This is an.... interesting guess, but it's totally wrong. After this point, I give up on the GPT3 for a while, and keep poking at the generated hex and the actual hex in python until I find similarities. Eventually, I figure it out: I actually do convert both hexes to literal binary, and search from a binary substring of one in the other. I discover that there is an offset of 2 bits.

I just edit the code manually, compensating for the offset by dividing the bigint by 4:

```javascript
function bafyToHex(bafyString) {
  // Create a lookup table for the base32 alphabet
  var alphabet = 'abcdefghijklmnopqrstuvwxyz234567';
  var base = alphabet.length;
  var lookupTable = {};
  alphabet.split('').forEach(function(char, i) {
    lookupTable[char] = i;
  });

  // Decode the base32-encoded string into a big integer
  var bigInt = bafyString.split('').reduce(function(acc, curr) {
    return acc * BigInt(base) + BigInt(lookupTable[curr]);
  }, BigInt(0)) / BigInt(4);

  // Convert the big integer into a hexadecimal string
  var hexString = bigInt.toString(16);

  return 'e30101701220' + hexString.slice(-64);
}
```

Because I already know what to do, I also just code the part that generates the entire calldata myself:

```
function bafyToCalldata(bafyString) {
    return (
        '0x304e6ade' +
        'ee6c4522aab0003e8d14cd40a6af439055fd2577951148c14b6cea9a53475835' +
        '0000000000000000000000000000000000000000000000000000000000000040' +
        '0000000000000000000000000000000000000000000000000000000000000026' +
        bafyToHex(bafyString) +
        '00000000000000000000000000000000000000000000000000000000'
    )
}
```

Anyway, then I switch to the next task: the portion of the Javascript that actually sends a transaction.
I go back to the GPT3.

Write a javascript function to send an Ethereum transaction with the `calldata` passed as input and the `to` address 0x4976fb03c32e5b8cfe2b6ccb31c09ba78ebaba41, without using any libraries

To write a JavaScript function that sends an Ethereum transaction with the `calldata` passed as input and the `to` address `0x4976fb03c32e5b8cfe2b6ccb31c09ba78ebaba41`, you can use the `web3.eth.sendTransaction` method from the Web3.js library. Here is an example of how to implement this function:

NOOOOO! I SAID NO LIBRARIES!!!!1!1!

I tell it what to use directly:

Don't use the web3.js library. Instead, use `window.ethereum.send`

To write a JavaScript function that sends an Ethereum transaction with the `calldata` passed as input and the `to` address `0x4976fb03c32e5b8cfe2b6ccb31c09ba78ebaba41`, without using the Web3.js library, you can use the `window.ethereum.send` method. Here is an example of how to implement this function:

```
function sendEthereumTransaction(calldata) {
    // Set the "to" address for the transaction
    const to = '0x4976fb03c32e5b8cfe2b6ccb31c09ba78ebaba41';

    // Set the transaction options
    const options = {
      to: to,
      data: calldata,
      gas: 3000000,
      gasPrice: '10000000000'
    };

    // Send the transaction
    window.ethereum.send('eth_sendTransaction', [options], function(error, result) {
      if (error) {
        console.error(error);
      } else {
        console.log(result);
      }
    });
}
```

This is more successful. Two errors though:

1. A `from` address actually is required.
2. You can't stick an integer into the `gas` field, you need a hex value.

Also, post EIP-1559, there really isn't much point in hard-coding a `gasPrice`. From here, I do the rest of the work myself.

```
function bafyToHex(bafyString) {
  // Create a lookup table for the base32 alphabet
  var alphabet = 'abcdefghijklmnopqrstuvwxyz234567';
  var base = alphabet.length;
  var lookupTable = {};
  alphabet.split('').forEach(function(char, i) {
    lookupTable[char] = i;
  });

  // Decode the base32-encoded string into a big integer
  var bigInt = bafyString.split('').reduce(function(acc, curr) {
    return acc * BigInt(base) + BigInt(lookupTable[curr]);
  }, BigInt(0)) / BigInt(4);

  // Convert the big integer into a hexadecimal string
  var hexString = bigInt.toString(16);
```

I ask the GPT-3 some minor questions: how to declare an `async` function, and what keyword to use in Twitter search to search only tweets that contain images (needed to write this post). It answers both flawlessly: do `async function functionName` to declare an async function, and use `filter:images` to filter for tweets that contain images.

Conclusions

The GPT-3 chatbot was helpful as a programming aid, but it also made plenty of mistakes. Ultimately, I was able to get past its mistakes quickly because I had lots of domain knowledge:

- I know that it was unlikely that browsers would have a builtin for base 58, which is a relatively niche format mostly used in the crypto world, and so I immediately got suspicious of its attempt to suggest `atob`
- I could eventually recall that the hash being all-lowercase means it's base 32 and not base 58
- I knew that the data in the Ethereum transaction had to encode the IPFS hash in *some* sensible way, which led me to eventually come up with the idea of checking bit offsets
- I know that a simple "correct" way to convert between base A and base B is to go through some abstract integer representation as an in-between, and that Javascript supported big integers.
- I knew about `window.ethereum.send`
- When I got the error that I was not allowed to put an integer into the `gas` field, I knew immediately that it was supposed to be hex.

At this point, AI is quite far from being a substitute for human programmers. In this particular case, it only sped me up by a little bit: I could have figured things out with Google eventually, and indeed in one or two places I *did* go back to googling. That said, it did introduce me to some coding patterns I had not seen before, and it wrote the base converter faster than I would have on my own. For the boilerplate operation of writing the Javascript to send a simple transaction, it did quite well.

That said, AI is improving quickly and I expect it to keep improving further and ironing out bugs like this over time.

Addendum: while writing the part of this post that involved more copy-paste than thinking, I put on my music playlist on shuffle. The first song that started playing was, coincidentally, [Basshunter's Boten Anna]("Anna The Bot").

What in the Ethereum application ecosystem excites me

2022 Dec 05

Special thanks to Matt Huang, Santi Siri and Tina Zhen for feedback and review

Ten, five, or even two years ago, my opinions on what Ethereum and blockchains can do for the world were very abstract. "This is a general-purpose technology, like C++", I would say; of course, it has specific properties like decentralization, openness and censorship resistance, but beyond that it's just too early to say which specific applications are going to make the most sense.

Today's world is no longer that world. Today, enough time has passed that there are few ideas that are completely unexplored: if something succeeds, it will *probably* be some version of something that has already been discussed in blogs and forums and conferences on multiple occasions. We've also come closer to identifying fundamental limits of the space. Many DAOs have had a fair chance with an enthusiastic audience willing to participate in them despite the inconveniences and fees, and many have underperformed. Industrial supply-chain applications [have not gone anywhere](). Decentralized Amazon on the blockchain has not happened. But it's also a world where we have seen genuine and growing adoption of a few key applications that are meeting people's real needs - and those are the applications that we need to focus on.

Hence my change in perspective: my excitement about Ethereum is now no longer based in the potential for undiscovered unknowns, but rather in a few specific categories of applications that are proving themselves already, and are only getting stronger. What are these applications, and which applications am I no longer optimistic about? That is what this post will be about.

1. Money: the first and still most important app

When I first visited Argentina in December last year, one of the experiences I remember well was walking around on Christmas Day, when almost everything is closed, looking for a coffee shop. After passing by about five closed ones, we finally found one that was open. When we walked in, the owner recognized me, and immediately showed me that he has ETH and other crypto-assets on his Binance account. We ordered tea and snacks, and we asked if we could pay in ETH. The coffee shop owner obliged, and showed me the QR code for his Binance deposit address, to which I sent about $20 of ETH from my Status wallet on my phone.

This was far from the most meaningful use of cryptocurrency that is taking place in the country. Others are using it to save money, transfer money internationally, make payments for large and important transactions, and much more. But even still, the fact that I randomly found a coffee shop and it happened to accept cryptocurrency showed the sheer reach of adoption. Unlike wealthy countries like the United States, where financial transactions are easy to make and 8% inflation is considered extreme, in [Argentina]() and [many]() other [countries]() around the world, links to global financial systems are more limited and extreme inflation is a [reality]() every day. Cryptocurrency often steps in as a lifeline.

In addition to Binance, there is also an increasing number of local exchanges, and you can see advertisements for them everywhere including at airports.

The one issue with my coffee transaction is that it did not really make pragmatic sense. The fee was high, about a third of the value of the transaction. The transaction took several minutes to confirm: I believe that at the time, Status did not yet support sending proper EIP-1559 transactions that more reliably confirm quickly. If, like many other Argentinian crypto users, I had simply had a Binance wallet, the transfer would have been free and instant.

A year later, however, the calculus is different. As a side effect of the Merge, transactions get included significantly more quickly and the chain has become more stable, making it safer to accept transactions after fewer confirmations. Scaling technology such as optimistic and ZK rollups is proceeding quickly. Social recovery and multisig wallets are becoming more practical with account abstraction. These trends will take years to play out as the technology develops, but progress is already being made. At the same time, **there is an important "push factor" driving interest in transacting on-chain: the FTX collapse**, which has reminded everyone, Latin Americans included, that even the most trustworthy-seeming centralized services may not be trustworthy after all.

Cryptocurrency in wealthy countries

In wealthy countries, the more extreme use cases around surviving high inflation and doing basic financial activities at all usually do not apply. But cryptocurrency still has significant value. As someone who has used it to make donations (to quite normal organizations in many countries), I can personally confirm that it is *far* more convenient than traditional banking. It's also valuable for industries and activities at risk of being deplatformed by payment processors - a category which includes many industries that are perfectly legal under most countries' laws.

There is also the important broader philosophical case for cryptocurrency as private money: the transition to a "cashless society" is being taken advantage of by many governments as an opportunity to introduce levels of financial surveillance that would be unimaginable 100 years ago. **Cryptocurrency is the *only* thing currently being developed that can realistically combine the benefits of digitalization with cash-like respect for personal privacy**.

But in either case, cryptocurrency is far from perfect. Even with all the technical, user experience and account safety problems solved, it remains a fact that cryptocurrency is volatile, and the volatility can make it difficult to use for savings and business. For that reason, we have...

Stablecoins

The value of stablecoins has been understood in the Ethereum community for a long time. Quoting a blog post from 2014:

> Over the past eleven months, Bitcoin holders have lost about 67% of their wealth and quite often the price moves up or down by as much as 25% in a single week. Seeing this concern, there is a growing interest in a simple question: can we get the best of both worlds? Can we have the full decentralization that a cryptographic payment network offers, but at the same time have a higher level of price stability, without such extreme upward and downward swings?

And indeed, stablecoins are very popular among precisely those users who are making pragmatic use of cryptocurrency today. That said, there is a reality that is not congenial to cypherpunk values today: the stablecoins that are most successful today are the centralized ones, mostly USDC, USDT and BUSD.

#	Coin	Price	1h	24h	7d	Mkt Cap
☆ 1	Bitcoin BTC	$16,906.10	1.0%	3.5%	4.7%	$324,950,635,810
☆ 2	Ethereum ETH	$1,280.66	1.2%	5.9%	12.8%	$154,281,907,530
☆ 3	Tether USDT	$1.01	0.7%	0.5%	0.7%	$65,891,847,384
☆ 4	BNB BNB	$299.04	1.3%	1.2%	12.3%	$48,841,350,953
☆ 5	USD Coin USDC	$1.00	0.4%	0.2%	0.4%	$43,475,130,147
☆ 6	Binance USD BUSD	$1.01	0.7%	0.4%	0.5%	$22,395,277,875

Top cryptocurrency market caps, data from CoinGecko, 2022-11-30. Three of the top six are centralized stablecoins.

Stablecoins issued on-chain have many convenient properties: they are open for use by anyone, they are resistant to the most large-scale and opaque forms of censorship (the issuer *can* blacklist and freeze addresses, but such blacklisting is transparent, and there are literal transaction fee costs associated with freezing each address), and they interact well with on-chain infrastructure (accounts, DEXes, etc). But it's not clear how long this state of affairs will last, and so there is a need to keep working on other alternatives.

I see the stablecoin design space as basically being split into three different categories: **centralized stablecoins**, **DAO-governed real-world-asset backed stablecoins** and **governance-minimized crypto-backed stablecoins**.

	Governance	Advantages	Disadvantages	Examples
Centralized stablecoins	Traditional legal entity	• Maximum efficiency • Easy to understand	Vulnerable to risks of a single issuer and a single jurisdiction	USDC, USDT, BUSD
DAO-governed RWA-backed stablecoins	DAO deciding on allowed collateral types and maximum per type	• Adds resilience by diversifying issuers and jurisdictions • Still somewhat capital efficient	Vulnerable to repeated issuer fraud or coordinated takedown	DAI
Governance-minimized crypto-backed stablecoin	Price oracle only	• Maximum resilience • No outside dependencies	• High collateral requirements • Limited scale • Sometimes needs negative interest rates	RAI, LUSD

From the user's perspective, the three types are arranged on a tradeoff spectrum between efficiency and resilience. USDC works today, and will almost certainly work tomorrow. But in the longer term, its ongoing stability depends on the macroeconomic and political stability of the United States, a continued US regulatory environment that supports making USDC available to everyone, and the trustworthiness of the issuing organization.

RAI, on the other hand, can survive all of these risks, but it has a negative interest rate: at the time of this writing, -6.7%. To make the system stable (so, not be vulnerable to collapse like LUNA), every holder of RAI must be matched by a holder of negative RAI (aka. a "borrower" or "CDP holder") who puts in ETH as collateral. This rate could be improved with more people engaging in arbitrage, holding negative RAI and balancing it out with positive USDC or even interest-bearing bank account deposits, but interest rates on RAI will always be lower than in a functioning banking system, and the possibility of negative rates, and the user experience headaches that they imply, will always be there.

The RAI model is ultimately ideal for the more pessimistic lunarpunk world: it avoids all connection to non-crypto financial systems, making it much more difficult to attack. Negative interest rates prevent it from being a convenient proxy for the dollar, but one way to adapt would be to embrace the disconnection: **a governance-minimized stablecoin could track some non-currency asset like a global average CPI index, and advertise itself as representing abstract "best-effort price stability"**. This would also have lower inherent regulatory risk, as such an asset would not be attempting to provide a "digital dollar" (or euro, or...).

DAO-governed RWA-backed stablecoins, if they can be made to work well, could be a happy medium. Such stablecoins could combine enough robustness, censorship resistance, scale and economic practicality to satisfy the needs of a large number of real-world crypto users. But making this work requires both real-world legal work to develop robust issuers, *and* a healthy dose of resilience-oriented DAO governance engineering.

In either case, *any* kind of stablecoin working well would be a boon for many kinds of currency and savings applications that are already concretely useful for millions of people today.

2. Defi: keep it simple

Decentralized finance is, in my view, a category that started off honorable but limited, turned into somewhat of an overcapitalized monster that relied on unsustainable forms of yield farming, and is now in the early stages of setting down into a stable medium, improving security and refocusing on a few applications that are particularly valuable. Decentralized stablecoins are, and probably forever will be, the most important defi product, but there are a few others that have an important niche:

- **Prediction markets**: these have been a niche but stable pillar of decentralized finance since the launch of Augur in 2015. Since then, they have quietly been growing in adoption. Prediction markets showed their value and their limitations in the 2020 US election, and this year in 2022, both crypto prediction markets like Polymarket and play-money markets like Metaculus are becoming more and more widely used. Prediction markets are valuable as an epistemic tool, and there is a genuine benefit from using cryptocurrency in making these markets more trustworthy and more globally accessible. I expect prediction markets to not make extreme multibillion-dollar splashes, but continue to steadily grow and become more useful over time.
- **Other synthetic assets**: the formula behind stablecoins can in principle be replicated to other real-world assets. Interesting natural candidates include major stock indices and real estate. The latter will take longer to get right due to the inherent heterogeneity and complexity of the space, but it could be valuable for precisely the same reasons. The main question is whether or not someone can create the right balance of decentralization and efficiency that gives users access to these assets at reasonable rates of return.
- **Glue layers for efficiently trading between other assets**: if there are assets on-chain that people want to use, including ETH, centralized or decentralized stablecoins, more advanced synthetic assets, or whatever else, there will be value in a layer that makes it easy for users to trade between them. Some users may want to hold USDC and pay transaction fees in USDC. Others may hold some assets, but want to be able to instantly convert to pay someone who wants to be paid in another asset. There is also space for using one asset as collateral to take out loans of another asset, though such projects are most likely to succeed and avoid leading to tears if they keep leverage *very* limited (eg. not more than 2x).

3. The identity ecosystem: ENS, SIWE, PoH, POAPs, SBTs

"Identity" is a complicated concept that can mean many things. Some examples include:

- **Basic authentication**: simply proving that action A (eg. sending a transaction or logging into a website) was authorized by some agent that has some identifier, such as an ETH address or a public key, without attempting to say anything else about who or what the agent is.
- **Attestations**: proving claims about an agent made by other agents ("Bob attests that he knows Alice", "the government of Canada attests that Charlie is a citizen")
- **Names**: establishing consensus that a particular human-readable name can be used to refer to a particular agent.
- **Proof of personhood**: proving that an agent is human, and guaranteeing that each human can only obtain one identity through the proof of personhood system (this is often done with attestations, so it's not an entirely separate category, but it's a hugely important special case)

For a long time, I have been bullish on *blockchain identity* but bearish on *blockchain identity platforms*. The use cases mentioned above are really important to many blockchain use cases, and blockchains are valuable for identity applications because of their institution-independent nature and the interoperability benefits that they provide. But what will not work is an attempt to create a centralized platform to achieve all of these tasks from scratch. What more likely will work is an organic approach, with many projects working on specific tasks that are individually valuable, and adding more and more interoperability over time.

And this is exactly what has happened since then. The Sign In With Ethereum (SIWE) standard allows users to log into (traditional) websites in much the same way that you can use Google or Facebook accounts to log into websites today. This is actually useful: it allows you to interact with a site without giving Google or Facebook access to your private information or the ability to take over or lock you out of your account. Techniques like social recovery could give users account recovery options in case they forget their password that are *much better* than what centralized corporations offer today. SIWE is supported by many applications today, including Blockscan chat, the end-to-end-encrypted email and notes service Skiff, and various blockchain-based alternative social media projects.

ENS lets users have usernames: I have vitalik.eth. Proof of Humanity and other proof-of-personhood systems let users prove that they are unique humans, which is useful in many applications including airdrops and governance. POAP (the "proof of attendance protocol", pronounced either "pope" or "poe-app" depending on whether you're a brave contrarian or a sheep) is a general-purpose protocol for issuing tokens that represent attestations: have you completed an educational course? Have you attended an event? Have you met a particular person? POAPs could be used both as an ingredient in a proof-of-personhood protocol and as a way to try to determine whether or not someone is a member of a particular community (valuable for governance or airdrops).

An NFC card that contains my ENS name, and allows you to receive a POAP verifying that you've met me. I'm not sure I **want** to create any further incentive for people to bug me really hard to get my POAP, but this seems fun and useful for other people.

Each of these applications are useful individually. But what makes them truly powerful is how well they compose with each other. When I log on to Blockscan chat, I sign in with Ethereum. This means that I am immediately visible as `vitalik.eth` (my ENS name) to anyone I chat with. In the future, to fight spam, Blockscan chat could "verify" accounts by looking at on-chain activity or POAPs. The lowest tier would simply be to verify that the account has sent or been the recipient in at least one on-chain transaction (as that requires paying fees). A higher level of verification could involve checking for balances of specific tokens, ownership of specific POAPs, a proof-of-personhood profile, or a meta-aggregator like Gitcoin Passport.

The network effects of these different services combine to create an ecosystem that provides some very powerful options for users and applications. An Ethereum-based Twitter alternative (eg. Farcaster) could use POAPs and other proofs of on-chain activity to create a "verification" feature that does not require conventional KYC, allowing anons to participate. Such platforms could create rooms that are gated to members of a particular community - or hybrid approaches where only community members can speak but anyone can listen. The equivalent of Twitter polls could be limited to particular communities.

Equally importantly, there are much more pedestrian applications that are relevant to simply helping people make a living: verification through attestations can make it easier for people to prove that they are trustworthy to get rent, employment or loans.

The big future challenge for this ecosystem is privacy. The status quo involves putting large amounts of information on-chain, which is something that is "fine until it's not", and eventually will become unpalatable if not outright risky to more and more people. There are ways to solve this problem by combining on-chain and off-chain information and making heavy use of ZK-SNARKs, but this is something that will actually need to be worked on; projects like Sismo and HeyAnon are an early start. Scaling is also a challenge, but scaling can be solved generically with rollups and perhaps validiums. Privacy cannot, and must be worked on intentionally for each application.

4. DAOs

"DAO" is a powerful term that captures many of the hopes and dreams that people have put into the crypto space to build more democratic, resilient and efficient forms of governance. It's also an incredibly broad term whose meaning has evolved a lot over the years. Most generally, a DAO is a smart contract that is meant to represent a structure of ownership or control over some asset or process. But this structure could be anything, from the lowly multisig to highly sophisticated multi-chamber governance mechanisms like those proposed for the Optimism Collective. Many of these structures work, and many others cannot, or at least are very mismatched to the goals that they are trying to achieve.

There are two questions to answer:

1. What kinds of governance structures make sense, and for what use cases?
2. Does it make sense to implement those structures as a DAO, or through regular incorporation and legal contracts?

A particular subtlety is that the word "decentralized" is sometimes used to refer to both: a *governance structure* is decentralized if its decisions depend on decisions taken from a large group of participants, and an *implementation* of a governance structure is decentralized if it is built on a decentralized structure like a blockchain and is not dependent on any single nation-state legal system.

Decentralization for robustness

One way to think about the distinction is: **decentralized governance structure protects against attackers on the inside, and a decentralized implementation protects against powerful attackers on the outside ("censorship resistance")**.

First, some examples:

	Higher need for protection from inside	Lower need for protection from inside
Higher need for protection from outside	Stablecoins	The Pirate Bay, Sci-Hub
Lower need for protection from outside	Regulated financial institutions	Regular businesses

The Pirate Bay and Sci-Hub are important case studies of something that is censorship-resistant, but does not need decentralization. Sci-Hub is largely run by one person, and if some part of Sci-Hub infrastructure gets taken down, she can simply move it somewhere else. The Sci-Hub URL has changed many times over the years. The Pirate Bay is a hybrid: it relies on BitTorrent, which *is* decentralized, but the Pirate Bay itself is a centralized convenience layer on top.

The difference between these two examples and blockchain projects is that they do not attempt to protect their users against the platform itself. If Sci-Hub or The Pirate Bay wanted to harm their users, the worst they could do is either serve bad results or shut down - either of which would only cause minor inconvenience until their users switch to other alternatives that would inevitably pop up in their absence. They could also publish user IP addresses, but even if they did that the total harm to users would still be much lower than, say, stealing all the users' funds.

Stablecoins are *not* like this. Stablecoins are trying to create stable credibly neutral global commercial infrastructure, and this demands both lack of dependence on a single centralized actor on the outside *and* protection against attackers from the inside. If a stablecoin's governance is poorly designed, an attack on the governance could steal billions of dollars from users.

At the time of this writing, MakerDAO has $7.8 billion in collateral, over 17x the market cap of the profit-taking token, MKR. Hence, if governance was up to MKR holders with no safeguards, someone could buy up half the MKR, use that to manipulate the price oracles, and steal a large portion of the collateral for themselves. In fact, this actually happened with a smaller stablecoin! It hasn't happened to MKR yet largely because the MKR holdings are still fairly concentrated, with the majority of the MKR held by a fairly small group that would not be willing to sell because they believe in the project. This is a fine model to get a stablecoin started, but not a good one for the long term. Hence, making decentralized stablecoins work long term requires innovating in decentralized governance that does not have these kinds of flaws.

Two possible directions include:

- Some kind of non-financialized governance, or perhaps a bicameral hybrid where decisions need to be passed not just by token holders but also by some other class of user (eg. the Optimism Citizens' House or stETH holders as in the Lido two-chamber proposal)
- Intentional friction, making it so that certain kinds of decisions can only take effect after a delay long enough that users can see that something is going wrong and escape the system.

There are many subtleties in making governance that effectively optimizes for robustness. If the system's robustness depends on pathways that are only activated in extreme edge cases, the system may even want to intentionally test those pathways once in a while to make sure that they work - much like the once-every-20-years rebuilding of Ise Jingu. This aspect of decentralization for robustness continues to require more careful thought and development.

Decentralization for efficiency

Decentralization for efficiency is a different school of thought: **decentralized governance structure is valuable because it can incorporate opinions from more diverse voices at different scales, and decentralized implementation is valuable because it can sometimes be more efficient and lower cost than traditional legal-system-based approaches**.

This implies a different *style* of decentralization. Governance decentralized for robustness emphasizes having a large number of decision-makers to ensure *alignment* with a pre-set goal, and *intentionally makes pivoting more difficult*. Governance decentralized for efficiency preserves the ability to act rapidly and pivot if needed, but tries to move decisions away from the top to avoid the organization becoming a sclerotic bureaucracy.
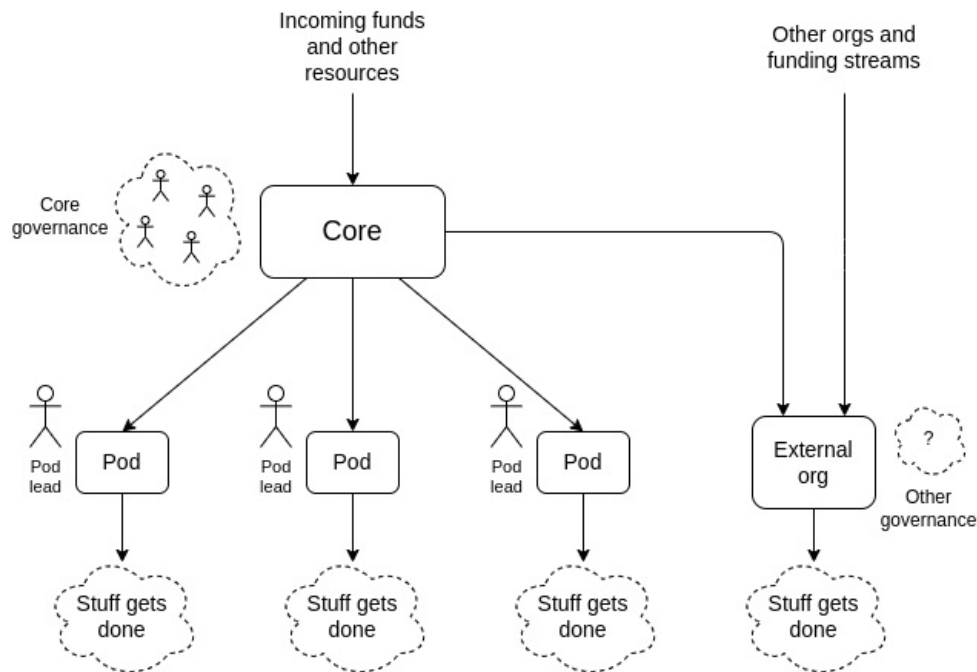
Pod-based governance in Ukraine DAO. This style of governance improves efficiency by maximizing autonomy.

Decentralized implementations designed for robustness and decentralized implementations designed for efficiency are in one way similar: they both just involve putting assets into smart contracts. But decentralized implementations designed for efficiency are going to be much simpler: just a basic multisig will generally suffice.

It's worth noting that "decentralizing for efficiency" is a weak argument for large-scale projects in the same wealthy country. But it's a stronger argument for very-small-scale projects, highly internationalized projects, and projects located in countries with inefficient institutions and weak rule of law. Many applications of "decentralizing for efficiency" probably *could* also be done on a central-bank-run chain run by a stable large country; I suspect that both decentralized approaches and centralized approaches are good enough, and it's the path-dependent question of which one becomes viable first that will determine which approach dominates.

Decentralization for interoperability

This is a fairly boring class of reasons to decentralize, but it's still important: **it's easier and more secure for on-chain things to interact with other on-chain things, than with off-chain systems that would inevitably require an (attackable) bridge layer**.

If a large organization running on direct democracy holds 10,000 ETH in its reserves, that would be a decentralized governance decision, but it would not be a decentralized implementation: in practice, that country would have a few people managing the keys and that storage system could get attacked.

There is also a governance angle to this: if a system provides services to *other* DAOs that are not capable of rapid change, it is better for that system to itself be incapable of rapid change, to avoid "rigidity mismatch" where a system's dependencies break and that system's rigidity renders it unable to adapt to the break.

These three "theories of decentralization" can be put into a chart as follows:

	Why decentralize governance structure	Why decentralize implementation
Decentralization for robustness	Defense against inside threats (eg. SBF)	Defense against outside threats, and censorship resistance
Decentralization for efficiency	Greater efficiency from accepting input from more voices and giving room for	Smart contracts often more convenient than legal systems

	autonomy	
Decentralization for interoperability	To be rigid enough to be safe to use by other rigid systems	To more easily interact with other decentralized things

Decentralization and fancy new governance mechanisms

Over the last few decades, we've seen the development of a number of fancy new governance mechanisms:

- [Quadratic voting](#)
- [Futarchy](#)
- [Liquid democracy](#)
- Decentralized *conversation* tools like [Pol.is](#)

These ideas are an important part of the DAO story, and they can be valuable for both robustness *and* efficiency. The case for quadratic voting relies on a mathematical argument that it makes the exactly correct tradeoff between giving space for stronger preferences to outcompete weaker but more popular preferences and not weighting stronger preferences (or wealthy actors) *too much*. But people who have used it have found that it can improve robustness too. Newer ideas, like [pairwise matching](#), *intentionally* sacrifice mathematically provable optimality for robustness in situations where the mathematical model's assumptions break.

These ideas, in addition to more "traditional" centuries-old ideas around multicameral architectures and intentional indirection and delays, are going to be an important part of the story in making DAOs more effective, though they will also find value in improving the efficiency of traditional organizations.

Case study: Gitcoin Grants

We can analyze the different styles of decentralization through an interesting edge-case: Gitcoin Grants. Should Gitcoin Grants be an on-chain DAO, or should it just be a centralized org?

Here are some possible arguments for Gitcoin Grants to be a DAO:

- It holds and deals with cryptocurrency, because most of its users and funders are Ethereum users
- Secure quadratic funding is best done on-chain (see next section on blockchain voting, and [implementation of on-chain QF here](#)), so you reduce security risks if the result of the vote feeds into the system directly
- It deals with communities all around the world, and so benefits from being [credibly neutral](#) and not centered around a single country.
- It benefits from being able to give its users confidence that it will still be around in five years, so that public goods funders can start projects now and hope to be rewarded later.

These arguments lean toward decentralization for robustness and decentralization for interoperability of the superstructure, though the individual quadratic funding rounds are more in the "decentralization for efficiency" school of thought (the theory behind Gitcoin Grants is that quadratic funding is a more efficient way to fund public goods).

If the robustness and interoperability arguments did *not* apply, then it probably would have been better to simply run Gitcoin Grants as a regular company. But they do apply, and so Gitcoin Grants being a DAO makes sense.

There are plenty of other examples of this kind of argument applying, both for DAOs that people increasingly rely on for their day-to-day lives, and for "meta-DAOs" that provide services to *other* DAOs:

- Proof of humanity
- [Kleros](#)
- [Chainlink](#)
- Stablecoins
- Blockchain layer 2 protocol governance

I don't know enough about all of these systems to testify that they all *do* optimize for decentralization-for-robustness enough to satisfy my standards, but hopefully it should be obvious by

now that they *should*.

The main thing that *does not* work well are DAOs that require pivoting ability that is in conflict with robustness, and that do not have a sufficient case to "decentralize for efficiency". Large-scale companies that mainly interface with US users would be one example. When making a DAO, the first thing is to determine whether or not it is worth it to structure the project as a DAO, and the second thing is to determine whether it's targeting robustness or efficiency: if the former, deep thought into governance *design* is also required, and if the latter, then either it's innovating on governance via mechanisms like quadratic funding, or it should just be a multisig.

5. Hybrid applications

There are many applications that are not entirely on-chain, but that take advantage of both blockchains and other systems to improve their trust models.

Voting is an excellent example. High assurances of censorship resistance, auditability and privacy are all required, and systems like MACI effectively combine blockchains, ZK-SNARKs and a limited centralized (or M-of-N) layer for scalability and coercion resistance to achieve all of these guarantees. Votes are published to the blockchain, so users have a way independent of the voting system to ensure that their votes get included. But votes are encrypted, preserving privacy, and a ZK-SNARK-based solution is used to ensure that the final result is the correct computation of the votes.
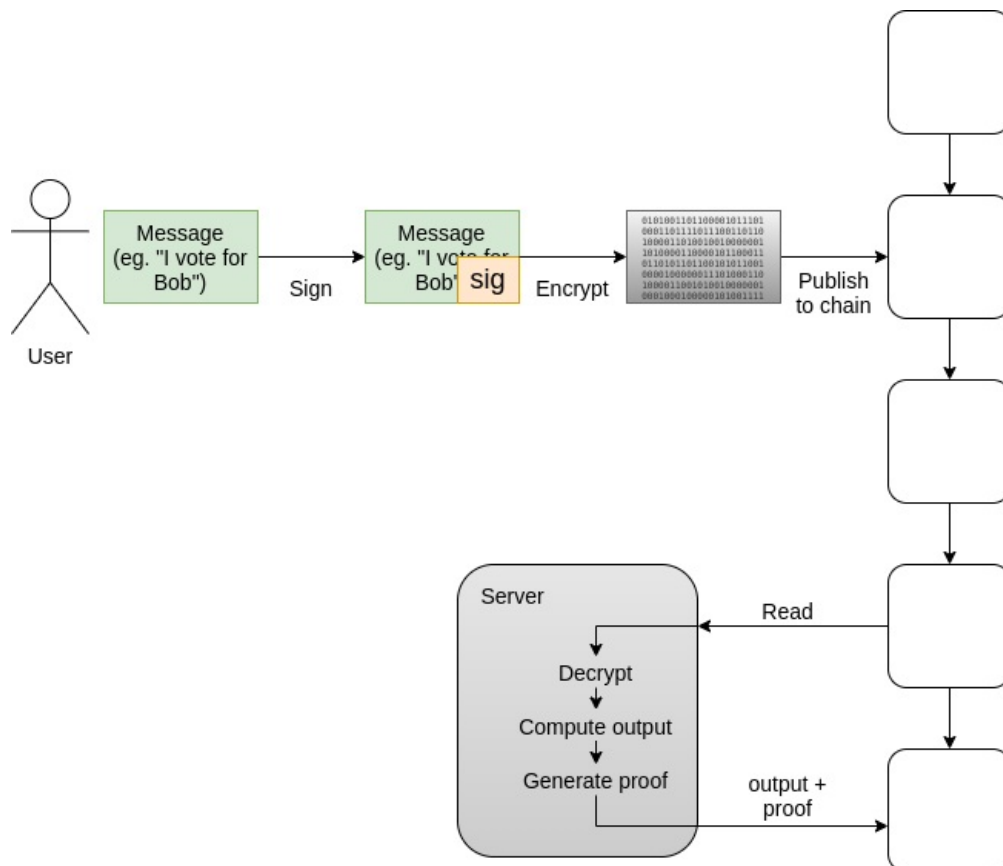
Diagram of how MACI works, combining together blockchains for censorship resistance, encryption for privacy, and ZK-SNARKs to ensure the result is correct without compromising on the other goals.

Voting in existing national elections is already a high-assurance process, and it will take a long time before countries and citizens are comfortable with the security assurances of any electronic ways to vote, blockchain or otherwise. But technology like this *can* be valuable very soon in two other places:

1. Increasing the assurance of voting processes that already happen electronically today (eg. social media votes, polls, petitions)

2. Creating new forms of voting that allow citizens or members of groups to give rapid feedback, and baking high assurance into those from the start

Going beyond voting, there is an entire field of potential "auditable centralized services" that could be well-served by some form of hybrid off-chain validium architecture. The easiest example of this is proof of solvency for exchanges, but there are plenty of other possible examples:

- Government registries
- Corporate accounting
- Games (see Dark Forest for an example)
- Supply chain applications
- Tracking access authorization
- ...

As we go further down the list, we get to use cases that are lower and lower value, but it is important to remember that these use cases are also quite low cost. Validiums do not require publishing everything on-chain. Rather, they can be simple wrappers around existing pieces of software that maintain a Merkle root (or other commitment) of the database and occasionally publish the root on-chain along with a SNARK proving that it was updated correctly. This is a strict improvement over existing systems, because it opens the door for cross-institutional proofs and public auditing.

So how do we get there?

Many of these applications are being built today, though many of these applications are seeing only limited usage because of the limitations of present-day technology. Blockchains are not scalable, transactions until recently took a fairly long time to reliably get included on the chain, and present-day wallets give users an uncomfortable choice between low convenience and low security. In the longer term, many of these applications will need to overcome the specter of privacy issues.

These are all problems that can be solved, and there is a strong drive to solve them. The FTX collapse has shown many people the importance of truly decentralized solutions to holding funds, and the rise of ERC-4337 and account abstraction wallets gives us an opportunity to create such alternatives. Rollup technology is rapidly progressing to solve scalability, and transactions already get included much more quickly on-chain than they did three years ago.

But what is also important is to be intentional about the application ecosystem itself. Many of the more stable and boring applications do not get built because there is less excitement and less short-term profit to be earned around them: the LUNA market cap got to over $30 billion, while stablecoins striving for robustness and simplicity often get largely ignored for years. Non-financial applications often have no hope of earning $30 billion because they do not have a token at all. But it is these applications that will be most valuable for the ecosystem in the long term, and that will bring the most lasting value to both their users and those who build and support them.

Having a safe CEX: proof of solvency and beyond

2022 Nov 19

Special thanks to Balaji Srinivasan, and Coinbase, Kraken and Binance staff for discussion.

Every time a major centralized exchange blows up, a common question that comes up is whether or not we can use cryptographic techniques to solve the problem. Rather than relying solely on "fiat" methods like government licenses, auditors and examining the corporate governance and the backgrounds of the individuals running the exchange, exchanges could create cryptographic *proofs* that show that the funds they hold on-chain are enough to cover their liabilities to their users.

Even more ambitiously, an exchange could build a system where it can't withdraw a depositor's funds at all without their consent. Potentially, we could explore the entire spectrum between the "don't be evil" aspiring-good-guy CEX and the "can't be evil", but for-now inefficient and privacy-leaking, on-chain DEX. This post will get into the history of attempts to move exchanges one or two steps closer to trustlessness, the limitations of these techniques, and some newer and more powerful ideas that rely on [ZK-SNARKs](#) and other advanced technologies.

Balance lists and Merkle trees: old-school proof-of-solvency

The earliest attempts by exchanges to try to cryptographically prove that they are not cheating their users go back quite far. In 2011, then-largest Bitcoin exchange MtGox proved that they had funds by sending [a transaction](#) that [moved 424242 BTC](#) to a pre-announced address. In 2013, [discussions started](#) on how to solve the other side of the problem: proving the total size of customers' deposits. If you prove that customers' deposits equal X ("**proof of liabilities**"), and prove ownership of the private keys of X coins ("**proof of assets**"), then you have a **proof of solvency**: you've proven the exchange has the funds to pay back all of its depositors.

The simplest way to prove deposits is to simply publish a list of (username, balance) pairs. Each user can check that their balance is included in the list, and anyone can check the full list to see that (i) every balance is non-negative, and (ii) the total sum is the claimed amount. Of course, this breaks privacy, so we can change the scheme a little bit: publish a list of (hash(username, salt), balance) pairs, and send each user privately their [salt](#) value. But even this leaks balances, and it leaks the pattern of changes in balances. The desire to preserve privacy [brings us](#) to the [next invention](#): **the Merkle tree technique**.
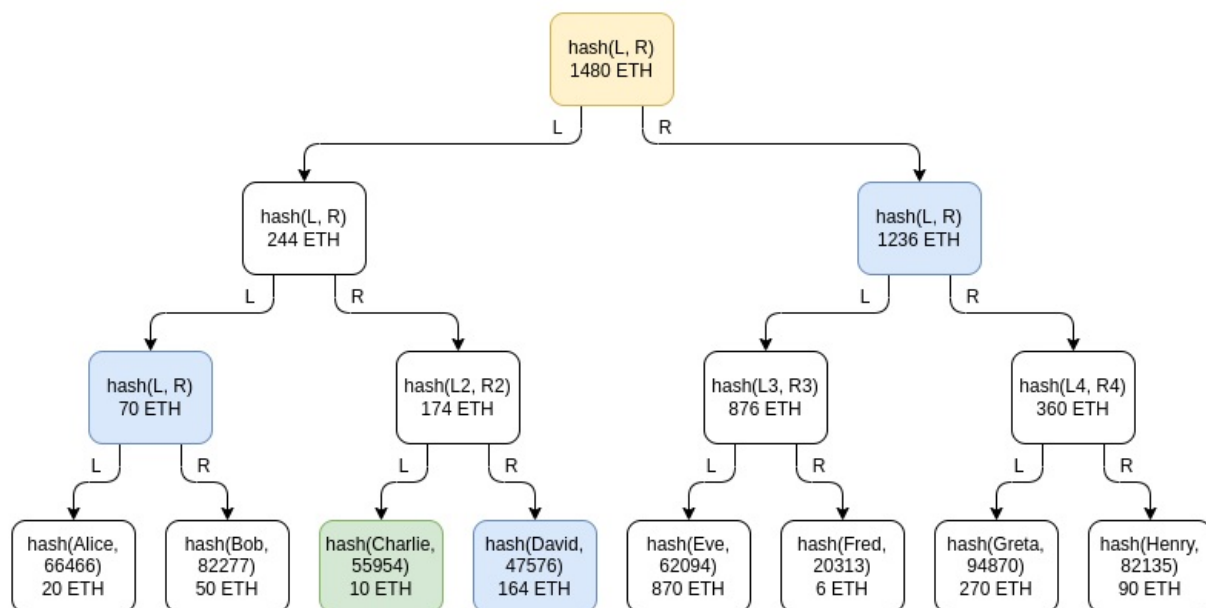
Green: Charlie's node. Blue: nodes Charlie will receive as part of his proof. Yellow: root node, publicly shown to everyone.

The Merkle tree technique consists of putting the table of customers' balances into a **Merkle sum tree**. In a Merkle sum tree, each node is a `(balance, hash)` pair. The bottom-layer leaf nodes represent the balances and salted username hashes of individual customers. In each higher-layer node, the balance is the sum of the two balances below, and the hash is the hash of the two nodes below. A Merkle sum proof, like a Merkle proof, is a "branch" of the tree, consisting of the sister nodes along the path from a leaf to the root.

The exchange would send each user a Merkle sum proof of their balance. The user would then have a guarantee that their balance is correctly included as part of the total. **A simple example code implementation can be found here**.

```
# The function for computing a parent node given two child nodes
def combine_tree_nodes(L, R):
    L_hash, L_balance = L
    R_hash, R_balance = R
    assert L_balance >= 0 and R_balance >= 0
    new_node_hash = hash(
        L_hash + L_balance.to_bytes(32, 'big') +
        R_hash + R_balance.to_bytes(32, 'big')
    )
    return (new_node_hash, L_balance + R_balance)

# Builds a full Merkle tree. Stored in flattened form where
# node i is the parent of nodes 2i and 2i+1
def build_merkle_sum_tree(user_table: "List[(username, salt, balance)]"):
    tree_size = get_next_power_of_2(len(user_table))
    tree = (
        [None] * tree size +
```

Privacy leakage in this design is much lower than with a fully public list, and it can be decreased further by shuffling the branches each time a root is published, but some privacy leakage is still there: Charlie learns that *someone* has a balance of 164 ETH, *some* two users have balances that add up to 70 ETH, etc. An attacker that controls many accounts could still potentially learn a significant amount about the exchange's users.

One important subtlety of the scheme is the possibility of *negative* balances: what if an exchange that has 1390 ETH of customer balances but only 890 ETH in reserves tries to make up the difference by adding a -500 ETH balance under a fake account somewhere in the tree? It turns out that this

possibility does not break the scheme, though this is the reason why we specifically need a Merkle sum tree and not a regular Merkle tree. Suppose that Henry is the fake account controlled by the exchange, and the exchange puts -500 ETH there:
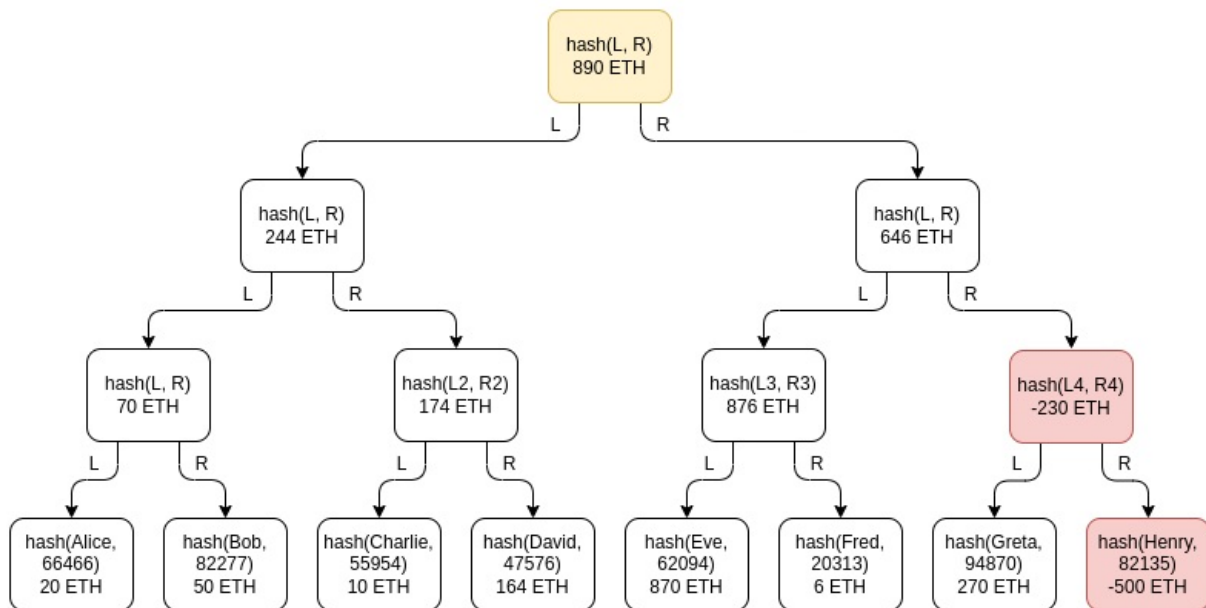
Greta's proof verification would fail: the exchange would have to give her Henry's -500 ETH node, which she would reject as invalid. Eve and Fred's proof verification would *also* fail, because the intermediate node above Henry has -230 total ETH, and so is also invalid! To get away with the theft, the exchange would have to hope that nobody in the entire right half of the tree checks their balance proof.

If the exchange can identify 500 ETH worth of users that they are confident will either not bother to check the proof, or will not be believed when they complain that they never received a proof, they could get away with the theft. But then the exchange could also just exclude those users from the tree and have the same effect. Hence, the Merkle tree technique is basically as good as a proof-of-liabilities scheme can be, if only achieving a proof of liabilities is the goal. But its privacy properties are still not ideal. You can go a little bit further by using Merkle trees in more clever ways, like making each satoshi or wei a separate leaf, but ultimately with more modern tech there are even better ways to do it.

Improving privacy and robustness with ZK-SNARKs

ZK-SNARKs are a powerful technology. ZK-SNARKs may be to cryptography what transformers are to AI: a general-purpose technology that is so powerful that it will completely steamroll a whole bunch of application-specific techniques for a whole bunch of problems that were developed in the decades prior. And so, of course, we can use ZK-SNARKs to greatly simplify and improve privacy in proof-of-liabilities protocols.

The simplest thing that we can do is put all users' deposits into a Merkle tree (or, even simpler, a KZG commitment), and use a ZK-SNARK to prove that all balances in the tree are non-negative and add up to some claimed value. If we add a layer of hashing for privacy, the Merkle branch (or KZG proof) given to each user would reveal *nothing* about the balance of any other user.

com(P)

$$Q = \frac{P - L[(\omega^4, hash(Charlie, 55954)),(\omega^5,10)]}{(X - \omega^4)(X - \omega^5)}$$

| hash(Alice, 66466) | 20 | hash(Bob, 82277) | 50 | hash(Charlie, 55954) | 10 | hash(David, 47576) | 164 | hash(Eve, 62094) | 870 | hash(Fred, 20313) | 6 | hash(Greta, 94870) | 270 | hash(Henry, 82135) | 90 |
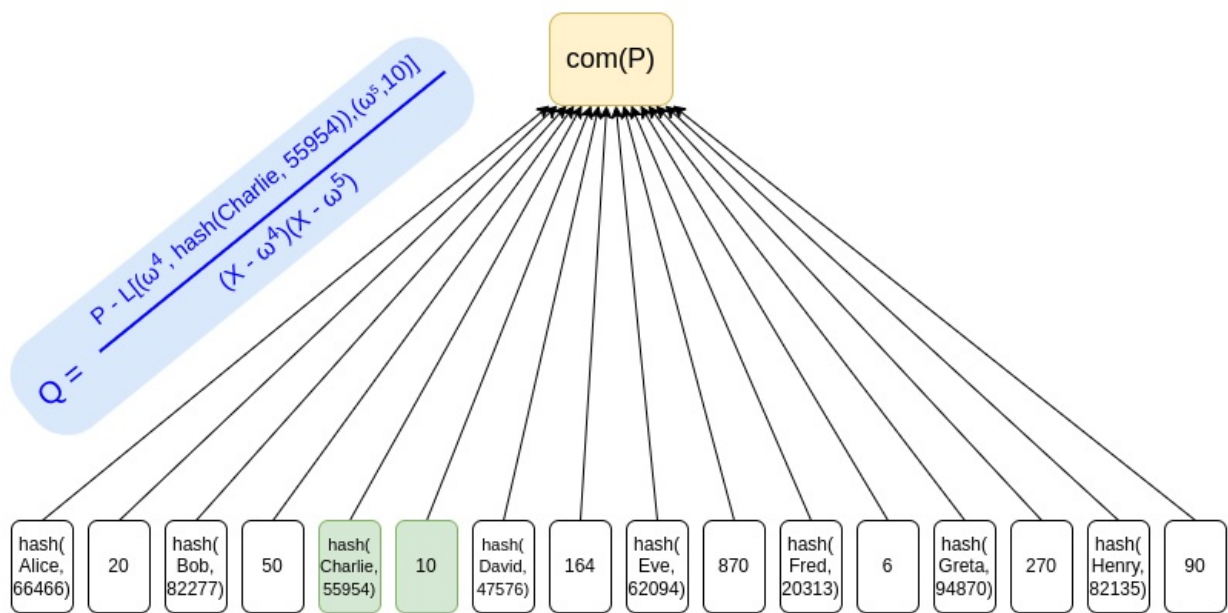
Using KZG commitments is one way to avoid privacy leakage, as there is no need to provide "sister nodes" as proofs, and a simple ZK-SNARK can be used to prove the sum of the balances and that each balance is non-negative.

We can prove the sum and non-negativity of balances in the above KZG with a special-purpose ZK-SNARK. Here is one simple example way to do this. We introduce an auxiliary polynomial $I(x)$, which "builds up the bits" of each balance (we assume for the sake of example that balances are under 2^{15}) and where every 16th position tracks a running total with an offset so that it sums to zero only if the actual total matches the declared total. If z is an order-128 root of unity, we might prove the equations:

$$I(z^{16x}) = 0$$

$$I(z^{16x + 14}) = P(\omega^{2x+1})$$

$$I(z^{i}) - 2*I(z^{i-1}) \in \{0, 1\}\ \ if\ \ i\ \ mod\ 16 \not \in \{0, 15\}$$

$$I(z^{16*x + 15}) = I(z^{16*x-1}) + I(z^{16*x+14}) - \frac{the\ declared\ total}{user\ count}$$

The first values of a valid setting for $I(x)$ would be `0 0 0 0 0 0 0 0 0 1 2 5 10 20 -165 0 0 0 0 0 0 0 0 0 1 3 6 12 25 50 -300 ...`

See here and here in my post on ZK-SNARKs for further explanation of how to convert equations like these into a polynomial check and then into a ZK-SNARK. This isn't an optimal protocol, but it does show how these days these kinds of cryptographic proofs are not that spooky!

With only a few extra equations, constraint systems like this can be adapted to more complex settings. For example, in a leverage trading system, an individual users having negative balances is acceptable but only if they have enough other assets to cover the funds with some collateralization margin. A SNARK could be used to prove this more complicated constraint, reassuring users that the exchange is not risking their funds by secretly exempting other users from the rules.

In the longer-term future, this kind of ZK proof of liabilities could perhaps be used not just for customer deposits at exchanges, but for lending more broadly. Anyone taking out a loan would put a record into a polynomial or a tree containing that loan, and the root of that structure would get published on-chain. This would let anyone seeking a loan ZK-prove to the lender that they have not yet taken out too many other loans. Eventually, legal innovation could even make loans that have been committed to in this way higher-priority than loans that have not. This leads us in exactly the same direction as one of the ideas that was discussed in the "Decentralized Society: Finding Web3's Soul" paper: a general notion of negative reputation or encumberments on-chain through some form of "soulbound tokens".

Proof of assets

The simplest version of proof of assets is the protocol that we saw above: to prove that you hold X coins, you simply move X coins around at some pre-agreed time or in a transaction where the data field contains the words "these funds belong to Binance". To avoid paying transaction fees, you could sign an off-chain message instead; both [Bitcoin](#) and [Ethereum](#) have standards for [off-chain](#) signed [messages](#).

There are two practical problems with this simple proof-of-assets technique:

- **Dealing with cold storage**
- **Collateral dual-use**

For safety reasons, most exchanges keep the great majority of customer funds in "cold storage": on offline computers, where transactions need to be signed and carried over onto the internet manually. Literal air-gapping is common: a cold storage setup that I used to use for personal funds involved a permanently offline computer generating a QR code containing the signed transaction, which I would scan from my phone. Because of the high values at stake, the security protocols used by exchanges are crazier still, and often involve using multi-party computation between several devices to further reduce the chance of a hack against a single device compromising a key. Given this kind of setup, making even a single extra message to prove control of an address is an expensive operation!

There are several paths that an exchange can take:

- **Keep a few public long-term-use addresses**. The exchange would generate a few addresses, publish a proof of each address *once* to prove ownership, and then use those addresses repeatedly. This is by far the simplest option, though it does add some constraints in how to preserve security and privacy.
- **Have many addresses, prove a few randomly**. The exchange would have many addresses, perhaps even using each address only once and retiring it after a single transaction. In this case, the exchange may have a protocol where from time to time a few addresses get randomly selected and must be "opened" to prove ownership. Some exchanges already do something like this with an auditor, but in principle this technique could be turned into a fully automated procedure.
- **More complicated ZKP options**. For example, an exchange could set all of its addresses to be 1-of-2 multisigs, where one of the keys is different per address, and the other is a blinded version of some "grand" emergency backup key stored in some complicated but very high-security way, eg. a 12-of-16 multisig. To preserve privacy and avoid revealing the entire set of its addresses, the exchange could even run a zero knowledge proof over the blockchain where it proves the total balance of all addresses on chain that have this format.

The other major issue is guarding against collateral dual-use. Shuttling collateral back and forth between each other to do proof of reserves is something that exchanges [could easily do](#), and would allow them to pretend to be solvent when they actually are not. Ideally, proof of solvency would be done in real time, with a proof that updates after every block. If this is impractical, the next best thing would be to coordinate on a fixed schedule between the different exchanges, eg. proving reserves at 1400 UTC every Tuesday.

One final issue is: **can you do proof-of-assets on fiat?** Exchanges don't just hold cryptocurrency, they also hold fiat currency within the banking system. Here, the answer is: yes, but such a procedure would inevitably rely on "fiat" trust models: the bank itself can attest to balances, auditors can attest to balance sheets, etc. Given that fiat is not cryptographically verifiable, this is the best that can be done within that framework, but it's still worth doing.

An alternative approach would be to cleanly separate between one entity that runs the exchange and deals with asset-backed stablecoins like USDC, and another entity (USDC itself) that handles the cash-in and cash-out process for moving between crypto and traditional banking systems. Because the "liabilities" of USDC are just on-chain ERC20 tokens, proof of liabilities comes "for free" and only proof of assets is required.

Plasma and validiums: can we make CEXes non-custodial?

Suppose that we want to go further: we don't want to just prove that the exchange *has the funds* to pay back its users. Rather, we want to **prevent the exchange from stealing users' funds completely**.

The first major attempt at this was **Plasma**, a scaling solution that was popular in Ethereum research circles in 2017 and 2018. Plasma works by splitting up the balance into a set of individual "coins", where each coin is assigned an index and lives in a particular position in the Merkle tree of a Plasma

block. Making a valid transfer of a coin requires putting a transaction into the correct position of a tree whose root gets published on-chain.
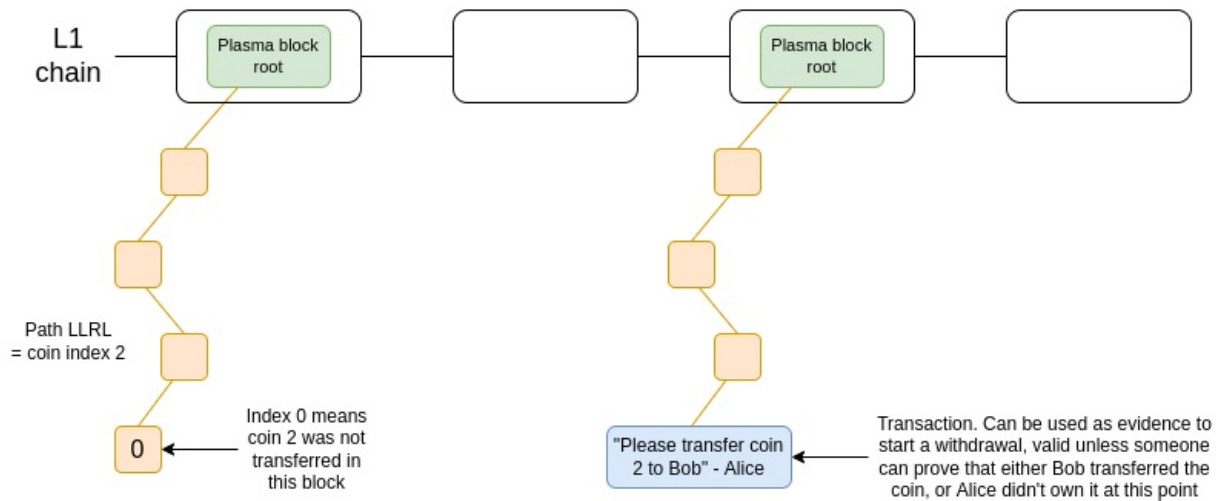
Oversimplified diagram of one version of Plasma. Coins are held in a smart contract that enforces the rules of the Plasma protocol at withdrawal time.

OmiseGo attempted to make a decentralized exchange based on this protocol, but since then they have pivoted to other ideas - as has, for that matter, Plasma Group itself, which is now the optimistic EVM rollup project Optimism.

It's not worth looking at the technical limitations of Plasma as conceived in 2018 (eg. proving coin defragmentation) as some kind of morality tale about the whole concept. Since the peak of Plasma discourse in 2018, ZK-SNARKs have become much more viable for scaling-related use cases, and as we have said above, ZK-SNARKs change everything.

The more modern version of the Plasma idea is what Starkware calls a validium: basically the same as a ZK-rollup, except where data is held off-chain. This construction could be used for a lot of use cases, conceivably anything where a centralized server needs to run some code and prove that it's executing code correctly. **In a validium, the operator has *no* way to steal funds, though depending on the details of the implementation some quantity of user funds could get *stuck* if the operator disappears.**

This is all really good: far from CEX vs DEX being a binary, it turns out that there is a whole spectrum of options, including various forms of hybrid centralization where you gain some benefits like efficiency but still have a lot of cryptographic guardrails preventing the centralized operator from engaging in most forms of abuses.
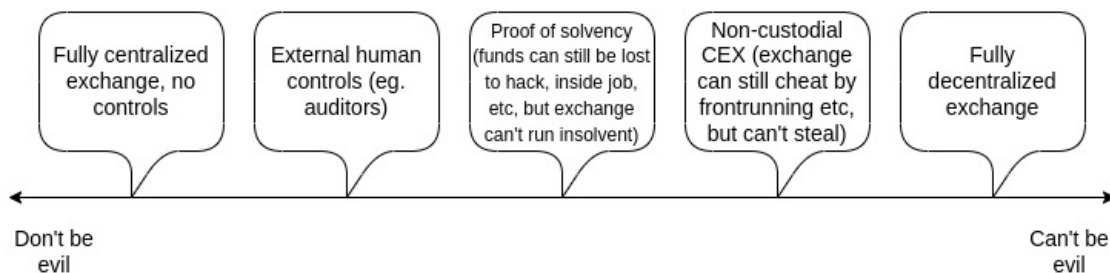
But it's worth getting to the fundamental issue with the right half of this design space: **dealing with user errors**. By far the most important type of error is: what if a user forgets their password, loses their devices, gets hacked, or otherwise loses access to their account?

Exchanges can solve this problem: first e-mail recovery, and if even that fails, more complicated forms of recovery through KYC. But to be able to solve such problems, the exchange needs to actually have control over the coins. In order to have the ability to recover user accounts' funds for good reasons, exchanges need to have power that could also be used to steal user accounts' funds for bad reasons. This is an unavoidable tradeoff.

The ideal long-term solution is to rely on self-custody, in a future where users have easy access to technologies such as [multisig and social recovery wallets](#) to help deal with emergency situations. But *in the short term*, there are two clear alternatives that have clearly distinct costs and benefits:

Option	Exchange-side risk	User-side risk
Custodial exchange (eg. Coinbase today)	User funds may be lost if there is a problem on the exchange side	Exchange can help recover account
Non-custodial exchange (eg. Uniswap today)	User can withdraw even if exchange acts maliciously	User funds may be lost if user screws up

Another important issue is cross-chain support: exchanges need to support many different chains, and systems like Plasma and validiums would need to have code written in different languages to support different platforms, and cannot be implemented at all on others (notably Bitcoin) in their current form. In the long-term future, this can hopefully be fixed with technological upgrades and standardization; in the short term, however, it's another argument in favor of custodial exchanges remaining custodial for now.

Conclusions: the future of better exchanges

In the short term, there are two clear "classes" of exchanges: custodial exchanges and non-custodial exchanges. Today, the latter category is just DEXes such as Uniswap, and in the future we may also see cryptographically "constrained" CEXes where user funds are held in something like a validium smart contract. We may also see half-custodial exchanges where we trust them with fiat but not cryptocurrency.

Both types of exchanges will continue to exist, and the easiest backwards-compatible way to improve the safety of custodial exchanges is to add proof of reserve. This consists of a combination of proof of assets and proof of liabilities. There are technical challenges in making good protocols for both, but we can and should go as far as possible to make headway in both, and open-source the software and processes as much as possible so that all exchanges can benefit.

In the longer-term future, my hope is that we move closer and closer to all exchanges being non-custodial, at least on the crypto side. Wallet recovery would exist, and there may need to be highly centralized recovery options for new users dealing with small amounts, as well as institutions that require such arrangements for legal reasons, but this can be done at the wallet layer rather than within the exchange itself. On the fiat side, movement between the traditional banking system and the crypto ecosystem could be done via cash in / cash out processes native to asset-backed stablecoins such as USDC. However, it will still take a while before we can fully get there.

The Revenue-Evil Curve: a different way to think about prioritizing public goods funding

2022 Oct 28

Special thanks to Karl Floersch, Hasu and Tina Zhen for feedback and review.

Public goods are an incredibly important topic in any large-scale ecosystem, but they are also one that is often surprisingly tricky to define. There is an economist definition of public goods - [goods that are non-excludable and non-rivalrous](), two technical terms that taken together mean that it's difficult to provide them through private property and market-based means. There is a layman's definition of public good: "anything that is good for the public". And there is a democracy enthusiast's definition of public good, which includes connotations of public participation in decision-making.

But more importantly, when the abstract category of non-excludable non-rivalrous public goods interacts with the real world, in almost any specific case there are all kinds of subtle edge cases that need to be treated differently. A park is a public good. But what if you add a $5 entrance fee? What if you fund it by auctioning off the right to have a statue of the winner in the park's central square? What if it's maintained by a semi-altruistic billionaire that enjoys the park for personal use, and designs the park around their personal use, but still leaves it open for anyone to visit?

This post will attempt to provide a different way of analyzing "hybrid" goods on the spectrum between private and public: **the revenue-evil curve**. We ask the question: what are the tradeoffs of different ways to monetize a given project, and how much good can be done by adding external subsidies to remove the pressure to monetize? This is far from a universal framework: it assumes a "mixed-economy" setting in a single monolithic "community" with a commercial market combined with subsidies from a central funder. But it can still tell us a lot about how to approach funding public goods in crypto communities, countries and many other real-world contexts today.

The traditional framework: excludability and rivalrousness

Let us start off by understanding how the usual economist lens views which projects are private vs public goods. Consider the following examples:

- Alice owns 1000 ETH, and wants to sell it on the market.
- Bob runs an airline, and sells tickets for a flight.
- Charlie builds a bridge, and charges a toll to pay for it.
- David makes and releases a podcast.
- Eve makes and releases a song.
- Fred invents a new and better cryptographic algorithm for making zero knowledge proofs.

We put these situations on a chart with two axes:

- **Rivalrousness**: to what extent does one person enjoying the good reduce another person's ability to enjoy it?
- **Excludability**: how difficult is it to prevent specific individuals, eg. those who do not pay, from enjoying the good?

Such a chart might look like this:
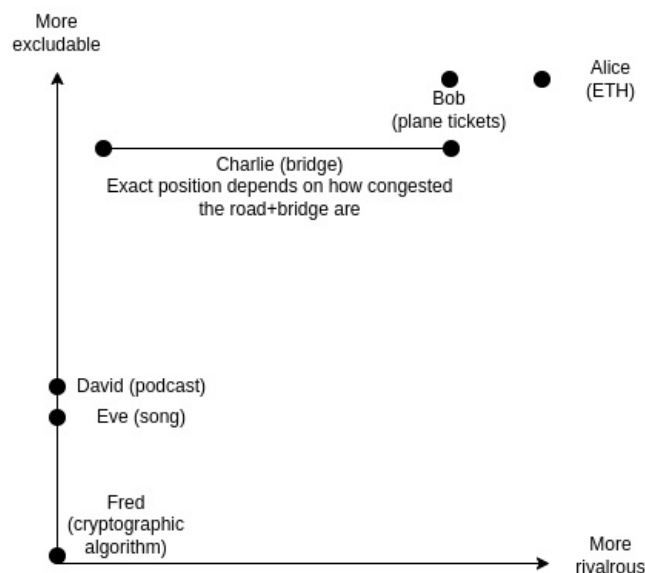
- Alice's ETH is completely excludable (she has total power to choose who gets her coins), and crypto coins are rivalrous (if one person owns a particular coin, no one else owns that same coin)
- Bob's plane tickets are excludable, but a tiny bit less rivalrous: there's a chance the plane won't be full.
- Charlie's bridge is a bit less excludable than plane tickets, because adding a gate to verify payment of tolls takes extra effort (so Charlie can exclude but it's costly, both to him and to users), and its rivalrousness depends on whether the road is congested or not.
- David's podcast and Eve's song are not rivalrous: one person listening to it does not interfere with another person doing the same. They're a little bit excludable, because you can make a paywall but people can circumvent the paywall.
- And Fred's cryptographic algorithm is close to not excludable at all: it needs to be open-source for people to trust it, and if Fred tries to patent it, the target user base (open-source-loving crypto users) may well refuse to use the algorithm and even cancel him for it.

This is all a good and important analysis. Excludability tells us whether or not you can fund the project by charging a toll as a business model, and rivalrousness tells us whether exclusion is a tragic waste or if it's just an unavoidable property of the good in question that if one person gets it another does not. But if we look at some of the examples carefully, especially the digital examples, we start to see that it misses a very important issue: **there are many business models available *other* than exclusion, and those business models have tradeoffs too**.

Consider one particular case: David's podcast versus Eve's song. In practice, a huge number of podcasts are released mostly or completely freely, but songs are more often gated with licensing and copyright restrictions. To see why, we need only look at how these podcasts are funded: sponsorships. Podcast hosts typically find a few sponsors, and talk about the sponsors briefly at the start or middle of each episode. Sponsoring songs is harder: you can't suddenly start talking about how awesome [Athletic Greens*](#) are in the middle of a love song, because come on, *it kills the vibe, man!*

Can we get beyond focusing solely on exclusion, and talk about monetization and the harms of different monetization strategies more generally? Indeed we can, and this is exactly what the revenue/evil curve is about.

The revenue-evil curve, defined

The revenue-evil curve of a product is a two-dimensional curve that plots the answer to the following question:

> **How much harm would the product's creator have to inflict on their potential users and the wider community to earn $N of revenue to pay for building the product?**

The word "evil" here is absolutely not meant to imply that no quantity of evil is acceptable, and that if you can't fund a project without committing evil you should not do it at all. Many projects make hard tradeoffs that hurt their customers and community in order to ensure sustainable funding, and often the value of the project existing at all greatly outweighs these harms. But nevertheless, the goal is to highlight that **there is a tragic aspect to many monetization schemes, and public goods funding can provide value by giving existing projects a financial cushion that enables them to avoid such sacrifices**.

Here is a rough attempt at plotting the revenue-evil curves of our six examples above:
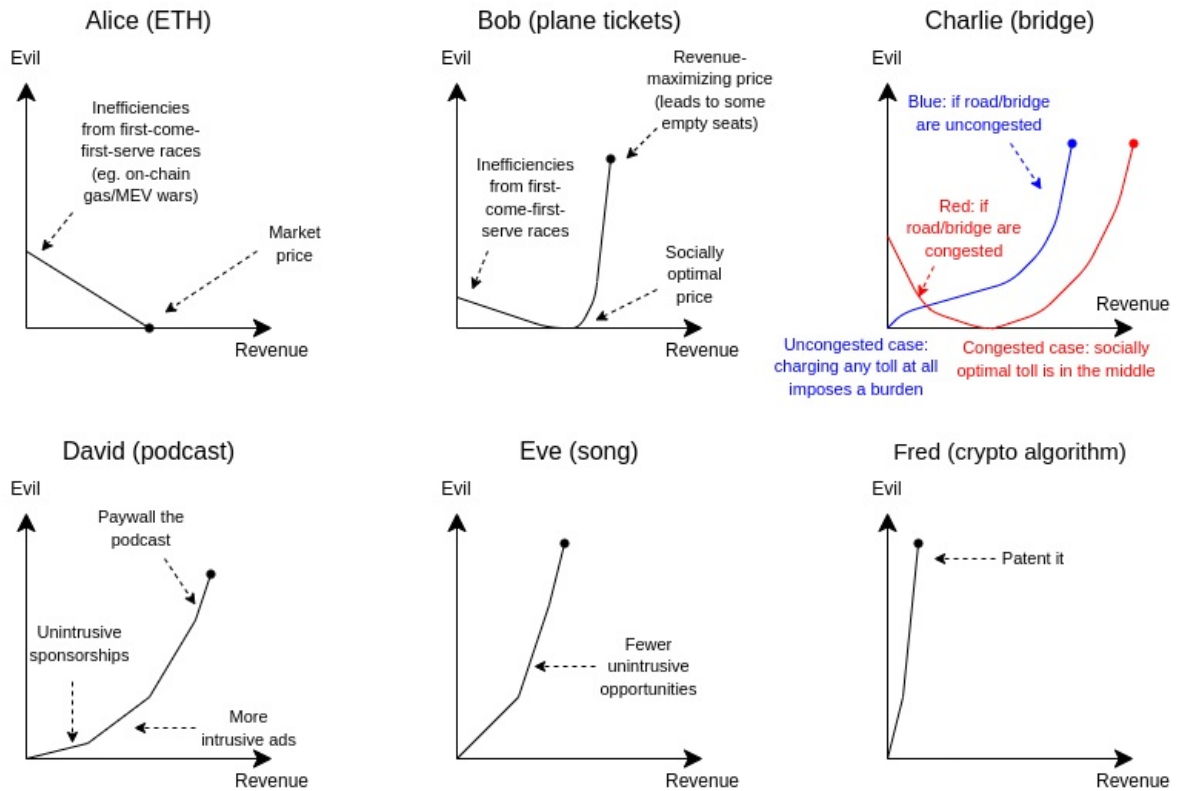
- For Alice, selling her ETH at market price is actually the *most* compassionate thing she could do. If she sells more cheaply, she will almost certainly create an on-chain gas war, trader HFT war, or other similarly value-destructive financial conflict between everyone trying to claim her coins the fastest. Selling above market price is not even an option: no one would buy.
- For Bob, the socially-optimal price to sell at is the highest price at which all tickets get sold out. If Bob sells below that price, tickets will sell out quickly and some people will not be able to get seats at all even if they really need them (underpricing may have a few countervailing benefits by giving opportunities to poor people, but it is far from the most efficient way to achieve that goal). Bob could also sell *above* market price and potentially earn a higher profit at the cost of selling fewer seats and (from the god's-eye perspective) needlessly excluding people.
- If Charlie's bridge and the road leading to it are uncongested, charging any toll at all imposes a burden and needlessly excludes drivers. If they are congested, low tolls help by reducing congestion and high tolls needlessly exclude people.
- David's podcast can monetize to some extent without hurting listeners much by adding advertisements from sponsors. If pressure to monetize increases, David would have to adopt more and more intrusive forms of advertising, and truly maxing out on revenue would require paywalling the podcast, a high cost to potential listeners.
- Eve is in the same position as David, but with fewer low-harm options (perhaps selling an NFT?). Especially in Eve's case, paywalling may well require actively participating in the legal apparatus of copyright enforcement and suing infringers, which carries further harms.
- Fred has even fewer monetization options. He could patent it, or potentially do exotic things like auction off the right to choose parameters so that hardware manufacturers that favor particular values would bid on it. All options are high-cost.

What we see here is that there are actually many kinds of "evil" on the revenue-evil curve:

- Traditional **economic deadweight loss from exclusion**: if a product is priced above marginal cost, mutually beneficial transactions that could have taken place do not take place
- **Race conditions**: congestion, shortages and other costs from products being too cheap.
- **"Polluting" the product** in ways that make it appealing to a sponsor, but is harmful to a (maybe small, maybe large) degree to listeners.
- Engaging in **offensive actions through the legal system**, which increases everyone's fear and need to spend money on lawyers, and has all kinds of hard-to-predict secondary chilling effects. This is particularly severe in the case of patenting.
- **Sacrificing on principles** highly valued by the users, the community and even the people working on the project itself.

In many cases, this evil is very context-dependent. Patenting is both extremely harmful and ideologically offensive within the crypto space and software more broadly, but this is less true in industries building physical goods: in physical goods industries, most people who realistically can create a derivative work of something patented are going to be large and well-organized enough to negotiate for a license, and capital costs mean that the need for monetization is much higher and hence maintaining purity is harder. To what extent advertisements are harmful depends on the advertiser and the audience: if the podcaster understands the audience *very* well, ads can even be helpful! Whether or not the possibility to "exclude" even exists depends on property rights.

But by talking about committing evil for the sake of earning revenue in general terms, we gain the ability to compare these situations against each other.

What does the revenue-evil curve tell us about funding prioritization?

Now, let's get back to the key question of why we care about what is a public good and what is not: funding prioritization. If we have a limited pool of capital that is dedicated to helping a community prosper, which things should we direct funding to? The revenue-evil curve graphic gives us a simple starting point for an answer: **direct funds toward those projects where the slope of the revenue-evil curve is the steepest**.

We should focus on projects where each $1 of subsidies, by reducing the pressure to monetize, most greatly reduces the evil that is unfortunately required to make the project possible. This gives us roughly this ranking:

- Top of the line are **"pure" public goods**, because often there aren't any ways to monetize them at all, or if there are, the economic or moral costs of trying to monetize are extremely high.
- Second priority is **"naturally" public but monetizable goods** that can be funded through commercial channels by tweaking them a bit, like songs or sponsorships to a podcast.
- Third priority is **non-commodity-like private goods** where social welfare is *already* optimized by charging a fee, but where profit margins are high or more generally there are opportunities to "pollute" the product to increase revenue, eg. by keeping accompanying software closed-source or refusing to use standards, and subsidies could be used to push such projects to make more pro-social choices on the margin.

Notice that the excludability and rivalrousness framework usually outputs similar answers: focus on non-excludable and non-rivalrous goods first, excludable goods but non-rivalrous second, and excludable and partially rivalrous goods last - and excludable and rivalrous goods never (if you have capital left over, it's better to just give it out as a UBI). There is a rough approximate mapping between revenue/evil curves and excludability and rivalrousness: higher excludability means lower slope of the revenue/evil curve, and rivalrousness tells us whether the bottom of the revenue/evil curve is zero or nonzero. But the revenue/evil curve is a much more general tool, which allows us to talk about tradeoffs of monetization strategies that go far beyond exclusion.

One practical example of how this framework can be used to analyze decision-making is Wikimedia donations. I personally have never donated to Wikimedia, because I've always thought that they could and should fund themselves without relying on limited public-goods-funding capital by just adding a few advertisements, and this would be only a small cost to their user experience and neutrality. Wikipedia admins, however, disagree; they even have a wiki page listing their arguments why they disagree.

We can understand this disagreement as a dispute over revenue-evil curves: I think Wikimedia's revenue-evil curve has a low slope ("ads are not that bad"), and therefore they are low priority for my

charity dollars; some other people think their revenue-evil curve has a high slope, and therefore they are high priority for their charity dollars.

Revenue-evil curves are an intellectual tool, NOT a good direct mechanism

One important conclusion that it is important NOT to take from this idea is that we should try to use revenue-evil curves directly as a way of prioritizing individual projects. There are severe constraints on our ability to do this because of **limits to monitoring**.

If this framework is widely used, projects would have an incentive to misrepresent their revenue-evil curves. Anyone charging a toll would have an incentive to come up with clever arguments to try to show that the world would be much better if the toll could be 20% lower, but because they're desperately under-budget, they just can't lower the toll without subsidies. Projects would have an incentive to be *more* evil in the short term, to attract subsidies that help them become less evil.

For these reasons, it is probably best to use the framework not as a way to allocate decisions directly, but to identify general principles for what kinds of projects to prioritize funding for. For example, the framework can be a valid way to determine how to prioritize whole industries or whole categories of goods. It can help you answer questions like: if a company is producing a public good, or is making pro-social but financially costly choices in the design of a not-quite-public good, do they deserve subsidies for that? But even here, it's better to treat revenue-evil curves as a mental tool, rather than attempting to precisely measure them and use them to make individual decisions.

Conclusions

Excludability and rivalrousness are important dimensions of a good, that have really important consequences for its ability to monetize itself, and for answering the question of how much harm can be averted by funding it out of some public pot. But especially once more complex projects enter the fray, these two dimensions quickly start to become insufficient for determining how to prioritize funding. Most things are not pure public goods: they are some hybrid in the middle, and there are many dimensions on which they could become more or less public that do not easily map to "exclusion".

Looking at the revenue-evil curve of a project gives us another way of measuring the statistic that really matters: how much harm can be averted by relieving a project of one dollar of monetization pressure? Sometimes, the gains from relieving monetization pressure are decisive: there just is no way to fund certain kinds of things through commercial channels, until you can find one single user that benefits from them enough to fund them unilaterally. Other times, commercial funding options exist, but have harmful side effects. Sometimes these effects are smaller, sometimes they are greater. Sometimes a small piece of an individual project has a clear tradeoff between pro-social choices and increasing monetization. And, still other times, projects just fund themselves, and there is no need to subsidize them - or at least, uncertainties and hidden information make it too hard to create a subsidy schedule that does more good than harm. It's always better to prioritize funding in order of greatest gains to smallest; and how far you can go depends on how much funding you have.

---

* I did not accept sponsorship money from [Athletic Greens](). But the podcaster Lex Fridman did. And no, I did not accept sponsorship money from Lex Fridman either. But maybe someone else did. Whatevs man, as long as we can keep getting podcasts funded so they can be free-to-listen without annoying people too much, it's all good, you know?

DAOs are not corporations: where decentralization in autonomous organizations matters

2022 Sep 20

Special thanks to Karl Floersch and Tina Zhen for feedback and review on earlier versions of this article.

Recently, there has been a lot of [discourse](#) around the [idea](#) that highly decentralized DAOs [do not work](#), and DAO governance [should](#) start to [more closely resemble](#) that of [traditional corporations](#) in order to remain competitive. The argument is always similar: highly decentralized governance is inefficient, and traditional corporate governance structures with boards, CEOs and the like evolved over hundreds of years to optimize for the goal of making good decisions and delivering value to shareholders in a changing world. DAO idealists are naive to assume that egalitarian ideals of decentralization can outperform this, when attempts to do this in the traditional corporate sector have had marginal success at best.

This post will argue why this position is often wrong, and offer a different and more detailed perspective about where different kinds of decentralization are important. In particular, I will focus on *three* types of situations where decentralization is important:

- **Decentralization for making better decisions in concave environments**, where pluralism and even naive forms of compromise are on average likely to outperform the kinds of coherency and focus that come from centralization.
- **Decentralization for censorship resistance**: applications that need to continue functioning while resisting attacks from powerful external actors.
- **Decentralization as credible fairness**: applications where DAOs are taking on nation-state-like functions like basic infrastructure provision, and so traits like predictability, robustness and neutrality are valued above efficiency.

Centralization is convex, decentralization is concave

See the original post: https://vitalik.ca/general/2020/11/08/concave.html

One way to categorize decisions that need to be made is to look at whether they are **convex** or **concave**. In a choice between A and B, we would first look not at the question of A vs B itself, but instead at a higher-order question: would you rather take a *compromise* between A and B or a *coin flip*? In expected utility terms, we can express this distinction using a graph:
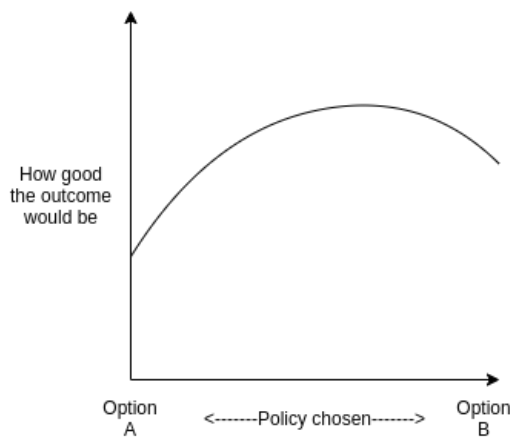
Concave worldview   Convex worldview
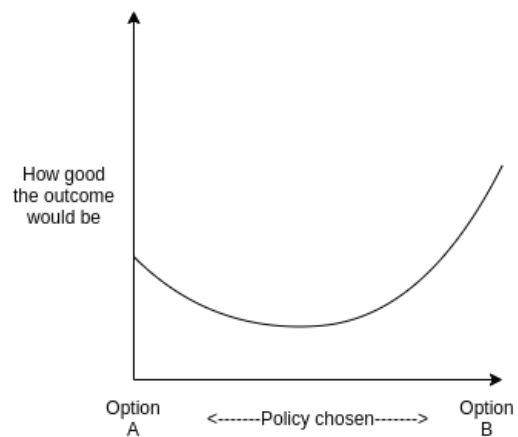
If a decision is concave, we would prefer a compromise, and if it's convex, we would prefer a coin flip. Often, we can answer the higher-order question of whether a compromise or a coin flip is better much more easily than we can answer the first-order question of A vs B itself.

Examples of convex decisions include:

- **Pandemic response**: a 100% travel ban may work at keeping a virus out, a 0% travel ban won't stop viruses but at least doesn't inconvenience people, but a 50% or 90% travel ban is the worst of both worlds.
- **Military strategy**: attacking on front A may make sense, attacking on front B may make sense, but splitting your army in half and attacking at both just means the enemy can easily deal with the two halves one by one
- **Technology choices in crypto protocols**: using technology A may make sense, using technology B may make sense, but some hybrid between the two often just leads to needless complexity and even adds risks of the two interfering with each other.

Examples of concave decisions include:

- **Judicial decisions**: an average between two independently chosen judgements is probably more likely to be fair, and less likely to be completely ridiculous, than a random choice of one of the two judgements.
- **Public goods funding**: usually, giving $X to each of two promising projects is more effective than giving $2X to one and nothing to the other. Having any money at all gives a much bigger boost to a project's ability to achieve its mission than going from $X to $2X does.
- **Tax rates**: because of quadratic deadweight loss mechanics, a tax rate of X% is often only a *quarter* as harmful as a tax rate of 2X%, and at the same time *more than half* as good at raising revenue. Hence, moderate taxes are better than a coin flip between low/no taxes and high taxes.

When decisions are convex, decentralizing the process of making that decision can easily lead to confusion and low-quality compromises. When decisions are concave, on the other hand, relying on the wisdom of the crowds can give *better* answers. In these cases, DAO-like structures with large amounts of diverse input going into decision-making can make a lot of sense. And indeed, people who see the world as a more concave place *in general* are more likely to see a need for decentralization in a wider variety of contexts.

Should VitaDAO and Ukraine DAO be DAOs?

Many of the more recent DAOs differ from earlier DAOs, like MakerDAO, in that whereas the earlier DAOs are organized around *providing infrastructure*, the newer DAOs are organized around *performing various tasks around a particular theme*. VitaDAO is a DAO funding early-stage longevity research, and UkraineDAO is a DAO organizing and funding efforts related to helping Ukrainian victims of war and supporting the Ukrainian defense effort. Does it make sense for these to be DAOs?

This is a nuanced question, and we can get a view of one possible answer by understanding the internal workings of UkraineDAO itself. Typical DAOs tend to "decentralize" by gathering large

amounts of capital into a single pool and using token-holder voting to fund each allocation. UkraineDAO, on the other hand, works by splitting its functions up into many pods, where each pod works as independently as possible. A top layer of governance can create new pods (in principle, governance can also fund pods, though so far funding has only gone to external Ukraine-related organizations), but once a pod is made and endowed with resources, it functions largely on its own. Internally, individual pods do have leaders and function in a more centralized way, though they still try to respect an ethos of personal autonomy.
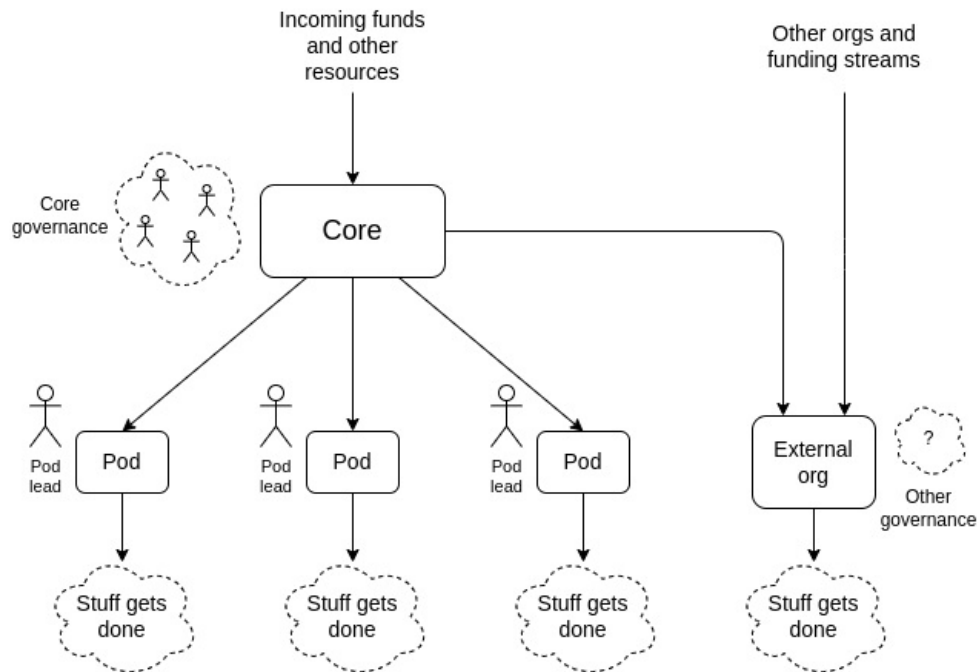
One natural question that one might ask is: **isn't this kind of "DAO" just rebranding the traditional concept of multi-layer hierarchy?** I would say this depends on the implementation: it's certainly possible to take this template and turn it into something that feels authoritarian in the same way stereotypical large corporations do, but it's also possible to use the template in a very different way.

Two things that can help ensure that an organization built this way will actually turn out to be meaningfully decentralized include:

1. A truly **high level of autonomy for pods**, where the pods accept resources from the core and are occasionally checked for alignment and competence if they want to keep getting those resources, but otherwise act entirely on their own and don't "take orders" from the core.
2. **Highly decentralized and diverse core governance**. This does not require a "governance token", but it does require broader and more diverse participation in the core. Normally, broad and diverse participation is a large tax on efficiency. But if (1) is satisfied, so pods are highly autonomous and the core needs to make fewer decisions, the effects of top-level governance being less efficient become smaller.

Now, how does this fit into the "convex vs concave" framework? Here, the answer is roughly as follows: **the (more decentralized) top level is concave, the (more centralized within each pod) bottom level is convex**. Giving a pod $X is generally better than a coin flip between giving it $0 and giving it $2X, and there isn't a large loss from having compromises or "inconsistent" philosophies guiding different decisions. But within each individual pod, having a clear opinionated perspective guiding decisions and being able to insist on many choices that have synergies with each other is much more important.

Decentralization and censorship resistance

The most often publicly cited reason for decentralization in crypto is censorship resistance: a DAO or

protocol needs to be able to function and defend itself despite external attack, including from large corporate or even state actors. This has already been [publicly talked about at length](#), and so deserves less elaboration, but there are still some important nuances.

Two of the most successful censorship-resistant services that large numbers of people use today are [The Pirate Bay](#) and [Sci-Hub](#). The Pirate Bay is a hybrid system: it's a search engine for BitTorrent, which is a highly decentralized network, but the search engine itself is centralized. It has a small core team that is dedicated to keeping it running, and it defends itself with the mole's strategy in whack-a-mole: when the hammer comes down, move out of the way and re-appear somewhere else. The Pirate Bay and Sci-Hub have both frequently changed domain names, relied on arbitrage between different jurisdictions, and used all kinds of other techniques. This strategy is centralized, but it has allowed them both to be successful *both* at defense and at product-improvement agility.

DAOs do not act like The Pirate Bay and Sci-Hub; DAOs act like BitTorrent. And **there is a reason why BitTorrent *does* need to be decentralized: it requires not just censorship resistance, but also long-term investment and reliability**. If BitTorrent got shut down once a year and required all its seeders and users to switch to a new provider, the network would quickly degrade in quality. Censorship resistance-demanding DAOs should also be in the same category: they should be providing a service that isn't just evading permanent censorship, but also evading mere instability and disruption. MakerDAO (and [the Reflexer DAO](#) which manages RAI) are excellent examples of this. A DAO running a decentralized search engine probably does not: you can just build a regular search engine and use Sci-Hub-style techniques to ensure its survival.

Decentralization as credible fairness

Sometimes, DAOs' primary concern is not a need to *resist* nation states, but rather a need to *take on some of the functions* of nation states. This often involves tasks that can be described as "maintaining basic infrastructure". Because governments have less ability to oversee DAOs, DAOs need to be structured to take on a greater ability to oversee *themselves*. And this requires decentralization.
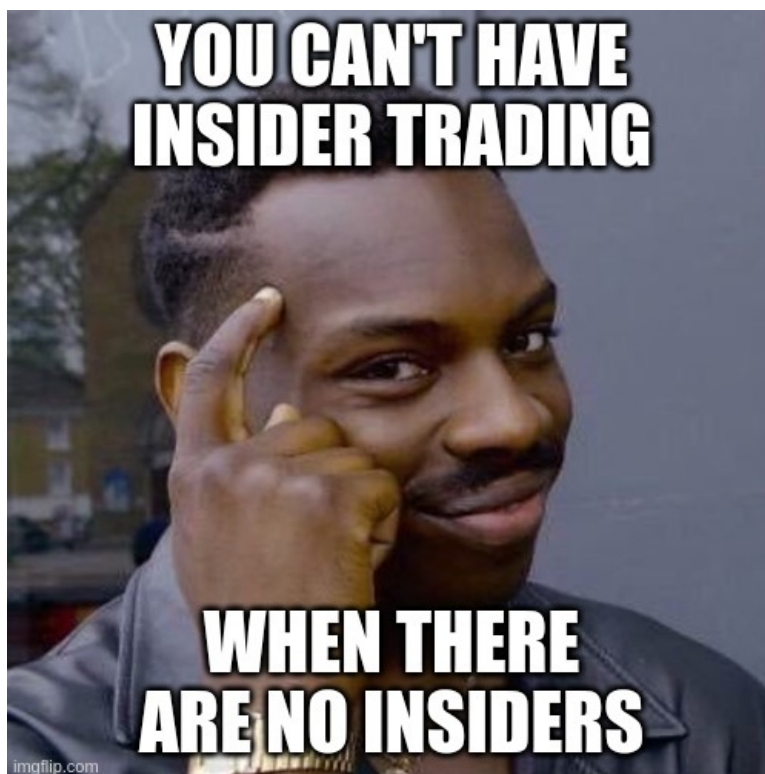
Of course, it's not actually possible to come anywhere close to eliminating hierarchy and inequality of information and decision-making power in its entirety etc etc etc, but what if we can get even 30% of the way there?

Consider three motivating examples: algorithmic stablecoins, the [Kleros court](#), and the [Optimism retroactive funding mechanism](#).

- **An algorithmic stablecoin DAO** is a system that uses on-chain financial contracts to create a crypto-asset whose price tracks some stable index, often but not necessarily the US dollar.
- **Kleros is a "[decentralized court](#)"**: a DAO whose function is to give rulings on arbitration questions such as "is this Github commit an acceptable submission to this on-chain bounty?"
- **Optimism's retroactive funding mechanism** is a component of the [Optimism DAO](#) which retroactively rewards projects that have provided value to the Ethereum and Optimism ecosystems.

In all three cases, there is a need to make subjective judgements, which cannot be done automatically through a piece of on-chain code. In the first case, the goal is simply to get reasonably accurate measurements of some price index. If the stablecoin tracks the US dollar, then you just need the ETH/USD price. If hyperinflation or some other reason to abandon the US dollar arises, the stablecoin DAO might need to manage a trustworthy on-chain CPI calculation. Kleros is all about making unavoidably subjective judgements on any arbitrary question that is submitted to it, including whether or not submitted questions should be [rejected for being "unethical"](#). Optimism's retroactive funding is tasked with one of the most open-ended subjective questions at all: what projects have done work that is the most useful to the Ethereum and Optimism ecosystems?

All three cases have an unavoidable need for "governance", and pretty robust governance too. In all cases, governance being attackable, from the outside or the inside, can easily lead to very big problems. Finally, the governance doesn't just need to *be* robust, it needs to *credibly convince* a large and untrusting public that it is robust.

The algorithmic stablecoin's Achilles heel: the oracle

Algorithmic stablecoins depend on oracles. In order for an on-chain smart contract to know whether to target the value of DAI to 0.005 ETH or 0.0005 ETH, it needs some mechanism to learn the (external-to-the-chain) piece of information of what the ETH/USD price is. And in fact, this "oracle" is the primary place at which an algorithmic stablecoin can be attacked.

This leads to a security conundrum: an algorithmic stablecoin cannot safely hold more collateral, and therefore cannot issue more units, than the market cap of its speculative token (eg. MKR, FLX...), because if it does, then it becomes profitable to buy up half the speculative token supply, use those tokens to control the oracle, and steal funds from users by feeding bad oracle values and liquidating them.

Here is a possible alternative design for a stablecoin oracle: [add a layer of indirection](#). Quoting the ethresear.ch post:

> We set up a contract where there are 13 "providers"; the answer to a query is the median of the answer returned by these providers. Every week, there is a vote, where the oracle token holders can replace one of the providers ...
>
> The security model is simple: if you trust the voting mechanism, you can trust the oracle output, unless 7 providers get corrupted at the same time. If you trust the current set of oracle providers, you can trust the output for at least the next six weeks, even if you completely do not trust the voting mechanism. Hence, if the voting mechanism gets corrupted, there will be able time for participants in any applications that depend on the oracle to make an orderly exit.

Notice the very un-corporate-like nature of this proposal. It involves *taking away* the governance's ability to act quickly, and intentionally spreading out oracle responsibility across a large number of participants. This is valuable for two reasons. First, it makes it harder for outsiders to attack the oracle, and for new coin holders to quickly take over control of the oracle. Second, it makes it harder for *the oracle participants themselves* to collude to attack the system. It also mitigates *oracle extractable value*, where a single provider might intentionally delay publishing to personally profit from a liquidation (in a multi-provider system, if one provider doesn't immediately publish, others soon will).

Fairness in Kleros

The "decentralized court" system Kleros is a really valuable and important piece of infrastructure for the Ethereum ecosystem: [Proof of Humanity](#) uses it, various "smart contract bug insurance" products use it, and many other projects plug into it as some kind of "adjudication of last resort".

Recently, there have been some public concerns about whether or not the platform's decision-making is fair. Some participants have made cases, trying to claim a payout from decentralized smart

contract insurance platforms that they argue they deserve. Perhaps the most famous of these cases is [Mizu's report on case #1170](). The case blew up from being a minor language intepretation dispute into a broader scandal because of the accusation that insiders to Kleros itself were making a coordinated effort to throw a large number of tokens to pushing the decision in the direction they wanted. A participant to the debate writes:

> The incentives-based decision-making process of the court ... is by all appearances being corrupted by a single dev with a very large (25%) stake in the courts.

Of course, this is but one side of one issue in a broader debate, and it's up to the Kleros community to figure out who is right or wrong and how to respond. But zooming out from the question of this individual case, what is important here is the the extent to which the *entire value proposition* of something like Kleros depends on it being able to convince the public that it is strongly protected against this kind of centralized manipulation. For something like Kleros to be trusted, it seems necessary that there should not be a single individual with a 25% stake in a high-level court. Whether through a more widely distributed token supply, or through more use of non-token-driven governance, a more credibly decentralized form of governance could help Kleros avoid such concerns entirely.

Optimism retro funding

Optimism's [retroactive founding round 1]() results were chosen by a quadratic vote among 24 "badge holders". Round 2 will likely use a larger number of badge holders, and the eventual goal is to move to a system where [a much larger body of citizens]() control retro funding allocation, likely through some multilayered mechanism involving sortition, subcommittees and/or delegation.

There have been some internal debates about whether to have more vs fewer citizens: should "citizen" *really* mean something closer to "senator", an expert contributor who deeply understands the Optimism ecosystem, should it be a position given out to *just about anyone* who has significantly participated in the Optimism ecosystem, or somewhere in between? **My personal stance on this issue has always been in the direction of more citizens, solving governance inefficiency issues with second-layer delegation instead of adding enshrined centralization into the governance protocol. One key reason for my position is the potential for insider trading and self-dealing issues**.

The Optimism retroactive funding mechanism has always been intended to be coupled with a *prospective* speculation ecosystem: public-goods projects that need funding *now* could sell "project tokens", and anyone who buys project tokens becomes eligible for a large retroactively-funded compensation later. But this mechanism working well depends crucially on the retroactive funding part working correctly, and is *very* vulnerable to the retroactive funding mechanism becoming corrupted. Some example attacks:

- If some group of people has decided how they will vote on some project, they can buy up (or if overpriced, short) its project token ahead of releasing the decision.
- If some group of people knows that they will later adjudicate on some specific project, they can buy up the project token early and then intentionally vote in its favor even if the project does not actually deserve funding.
- Funding deciders can accept bribes from projects.

There are typically three ways of dealing with these types of corruption and insider trading issues:

- *Retroactively punish* malicious deciders.
- Proactively *filter for higher-quality* deciders.
- Add *more* deciders.

The corporate world typically focuses on the first two, using financial surveillance and judicious penalties for the first and in-person interviews and background checks for the second. The decentralized world has less access to such tools: project tokens are likely to be tradeable anonymously, DAOs have at best limited recourse to external judicial systems, and the remote and online nature of the projects and the desire for global inclusivity makes it harder to do background checks and informal in-person "smell tests" for character. Hence, the decentralized world needs to put more weight on the third technique: distribute decision-making power among *more* deciders, so that each individual decider has less power, and so collusions are more likely to be whistleblown on and revealed.

Should DAOs learn more from corporate governance or

political science?

Curtis Yarvin, an American philosopher whose primary "big idea" is that corporations are much more effective and optimized than governments and so we should improve governments by making them look more like corporations (eg. by moving away from democracy and closer to monarchy), recently wrote an article expressing his thoughts on how DAO governance should be designed. Not surprisingly, his answer involves borrowing ideas from governance of traditional corporations. From his introduction:

> Instead the basic design of the Anglo-American limited-liability joint-stock company has remained roughly unchanged since the start of the Industrial Revolution—which, a contrarian historian might argue, might actually have been a Corporate Revolution. If the joint-stock design is not perfectly optimal, we can expect it to be nearly optimal.
>
> While there is a categorical difference between these two types of organizations—we could call them first-order (sovereign) and second-order (contractual) organizations—it seems that society in the current year has very effective second-order organizations, but not very effective first-order organizations.
>
> Therefore, we probably know more about second-order organizations. So, when designing a DAO, we should start from corporate governance, not political science.

Yarvin's post is very correct in identifying the key difference between "first-order" (sovereign) and "second-order" (contractual) organizations - in fact, that exact distinction is precisely the topic of the section in my own post above on credible fairness. However, Yarvin's post makes a big, and surprising, mistake immediately after, by immediately pivoting to saying that corporate governance is the better starting point for how DAOs should operate. The mistake is surprising because the logic of the situation seems to almost directly imply the exact opposite conclusion. **Because DAOs do not have a sovereign above them, and are often explicitly in the business of providing services (like currency and arbitration) that are typically reserved for sovereigns, it is precisely the design of sovereigns (political science), and not the design of corporate governance, that DAOs have more to learn from.**

To Yarvin's credit, the second part of his post *does* advocate an "hourglass" model that combines a decentralized alignment and accountability layer and a centralized management and execution layer, but this is already an admission that DAO design needs to learn at least as much from first-order orgs as from second-order orgs.

Sovereigns are inefficient and corporations are efficient for the same reason why number theory can prove very many things but abstract group theory can prove much fewer things: **corporations fail less and accomplish more because they can make more assumptions and have more powerful tools to work with**. Corporations can count on their local sovereign to stand up to defend them if the need arises, as well as to provide an external legal system they can lean on to stabilize their incentive structure. In a sovereign, on the other hand, the biggest challenge is often what to do when the incentive structure is under attack and/or at risk of collapsing entirely, with no external leviathan standing ready to support it.

Perhaps the greatest problem in the design of successful governance systems for sovereigns is what Samo Burja calls "the succession problem": how to ensure continuity as the system transitions from being run by one group of humans to another group as the first group retires. Corporations, Burja writes, often just don't solve the problem at all:

> Silicon Valley enthuses over "disruption" because we have become so used to the succession problem remaining unsolved within discrete institutions such as companies.

DAOs will need to solve the succession problem eventually (in fact, given the sheer frequency of the "get rich and retire" pattern among crypto early adopters, some DAOs have to deal with succession issues *already*). Monarchies and corporate-like forms often have a hard time solving the succession problem, because the institutional structure gets deeply tied up with the habits of one specific person, and it either proves difficult to hand off, or there is a very-high-stakes struggle over whom to hand it off to. More decentralized political forms like democracy have at least a theory of how smooth transitions can happen. Hence, I would argue that for this reason too, DAOs have more to learn from the more liberal and democratic schools of political science than they do from the governance of corporations.

Of course, DAOs will in some cases have to accomplish specific complicated tasks, and some use of corporate-like forms for accomplishing those tasks may well be a good idea. Additionally, DAOs need

to handle unexpected uncertainty. A system that was intended to function in a stable and unchanging way around one set of assumptions, when faced with an extreme and unexpected change to those circumstances, does need some kind of brave leader to coordinate a response. A prototypical example of the latter is stablecoins handling a US dollar collapse: what happens when a stablecoin DAO that evolved around the assumption that it's just trying to track the US dollar suddenly faces a world where the US dollar is no longer a viable thing to be tracking, and a rapid switch to some kind of CPI is needed?
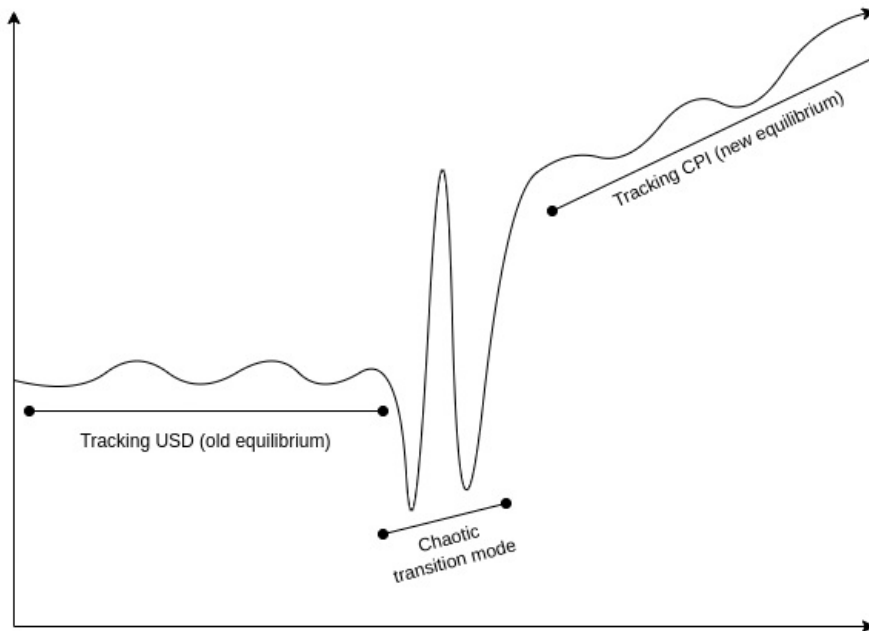
Stylized diagram of the internal experience of the RAI ecosystem going through an unexpected transition to a CPI-based regime if the USD ceases to be a viable reference asset.

Here, corporate governance-inspired approaches may seem better, because they offer a ready-made pattern for responding to such a problem: the founder organizes a pivot. But as it turns out, the history of political systems *also* offers a pattern well-suited to this situation, and one that covers the question of how to go back to a decentralized mode when the crisis is over: the Roman Republic custom of electing a dictator for a temporary term to respond to a crisis.

Realistically, we probably only need a small number of DAOs that look more like constructs from political science than something out of corporate governance. But those are the really important ones. A stablecoin does not need to be efficient; it must first and foremost be stable and decentralized. A decentralized court is similar. A system that directs funding for a particular cause - whether Optimism retroactive funding, VitaDAO, UkraineDAO or something else - is optimizing for a much more complicated purpose than profit maximization, and so an alignment solution other than shareholder profit is needed to make sure it keeps using the funds for the purpose that was intended.

By far the greatest number of organizations, even in a crypto world, *are* going to be "contractual" second-order organizations that ultimately lean on these first-order giants for support, and for these organizations, much simpler and leader-driven forms of governance emphasizing agility are often going to make sense. But this should not distract from the fact that the ecosystem would not survive without some *non-corporate* decentralized forms keeping the whole thing stable.

What kind of layer 3s make sense?

2022 Sep 17

Special thanks to Georgios Konstantopoulos, Karl Floersch and the Starkware team for feedback and review.

One topic that often re-emerges in layer-2 scaling discussions is the concept of "layer 3s". If we can build a layer 2 protocol that anchors into layer 1 for security and adds scalability on top, then surely we can scale even more by building a *layer 3* protocol that anchors into layer 2 for security and adds *even more scalability* on top of that?

A simple version of this idea goes: if you have a scheme that can give you quadratic scaling, can you stack the scheme on top of itself and get exponential scaling? Ideas like this include my 2015 scalability paper, the multi-layer scaling ideas in the Plasma paper, and many more. Unfortunately, such simple conceptions of layer 3s rarely quite work out that easily. There's always something in the design that's just not stackable, and can only give you a scalability boost once - limits to data availability, reliance on L1 bandwidth for emergency withdrawals, or many other issues.

Newer ideas around layer 3s, such as the framework proposed by Starkware, are more sophisticated: they aren't just stacking the same thing on top of itself, they're assigning the second layer and the third layer different purposes. Some form of this approach may well be a good idea - if it's done in the right way. This post will get into some of the details of what might and might not make sense to do in a triple-layered architecture.

Why you can't just keep scaling by stacking rollups on top of rollups

Rollups (see my longer article on them here) are a scaling technology that combines different techniques to address the two main scaling bottlenecks of running a blockchain: *computation* and *data*. Computation is addressed by either fraud proofs or SNARKs, which rely on a very small number of actors to process and verify each block, requiring everyone else to perform only a tiny amount of computation to check that the proving process was done correctly. These schemes, especially SNARKs, can scale almost without limit; you really can just keep making "a SNARK of many SNARKs" to scale *even more* computation down to a single proof.

Data is different. Rollups use a collection of compression tricks to reduce the amount of data that a transaction needs to store on-chain: a simple currency transfer decreases from ~100 to ~16 bytes, an ERC20 transfer in an EVM-compatible chain from ~180 to ~23 bytes, and a privacy-preserving ZK-SNARK transaction could be compressed from ~600 to ~80 bytes. About 8x compression in all cases. But rollups still need to make data available on-chain in a medium that users are guaranteed to be able to access and verify, so that users can independently compute the state of the rollup and join as provers if existing provers go offline. Data can be compressed once, but it cannot be compressed again - if it can, then there's generally a way to put the logic of the second compressor into the first, and get the same benefit by compressing once. Hence, "rollups on top of rollups" are not something that can actually provide large gains in scalability - though, as we will see below, such a pattern can serve other purposes.

So what's the "sane" version of layer 3s?

Well, let's look at what Starkware, in their post on layer 3s, advocates. Starkware is made up of very smart cryptographers who are actually sane, and so if they are advocating for layer 3s, their version will be much more sophisticated than "if rollups compress data 8x, then *obviously* rollups on top of rollups will compress data 64x".

Here's a diagram from Starkware's post:
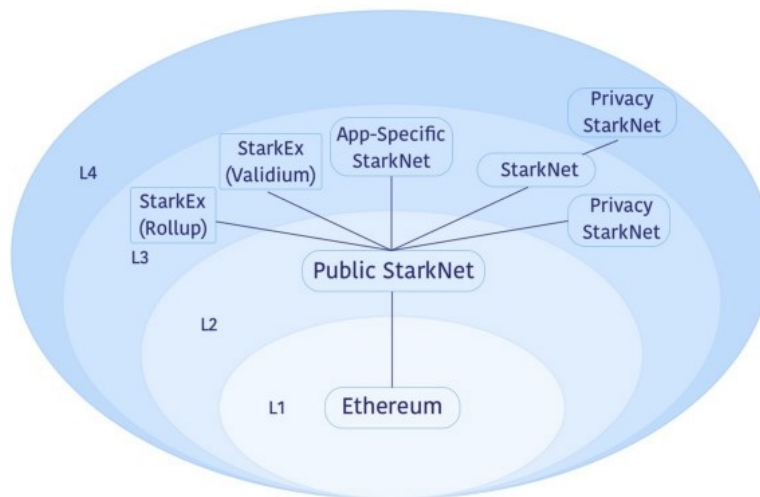
A few quotes:

> An example of such an ecosystem is depicted in Diagram 1. Its L3s include:
>
> - A StarkNet with Validium data availability, e.g., for general use by applications with extreme sensitivity to pricing.
> - App-specific StarkNet systems customized for better application performance, e.g., by employing designated storage structures or data availability compression.
> - StarkEx systems (such as those serving dYdX, Sorare, Immutable, and DeversiFi) with Validium or Rollup data availability, immediately bringing battle-tested scalability benefits to StarkNet.
> - Privacy StarkNet instances (in this example also as an L4) to allow privacy-preserving transactions without including them in public StarkNets.

We can compress the article down into three visions of what "L3s" are for:

1. **L2 is for scaling, L3 is for customized functionality, for example privacy.** In this vision there is no attempt to provide "scalability squared"; rather, there is one layer of the stack that helps applications scale, and then separate layers for customized functionality needs of different use cases.
2. **L2 is for general-purpose scaling, L3 is for customized scaling**. Customized scaling might come in different forms: specialized applications that use something other than the EVM to do their computation, rollups whose data compression is optimized around data formats for specific applications (including separating "data" from "proofs" and replacing proofs with a single SNARK per block entirely), etc.
3. **L2 is for trustless scaling (rollups), L3 is for weakly-trusted scaling (validiums)**. [Validiums](#) are systems that use SNARKs to verify computation, but leave data availability up to a trusted third party or committee. Validiums are in my view highly underrated: in particular, many "enterprise blockchain" applications may well actually be best served by a centralized server that runs a validium prover and regularly commits hashes to chain. Validiums have a lower grade of security than rollups, but can be vastly cheaper.

All three of these visions are, in my view, fundamentally reasonable. The idea that specialized data compression requires its own platform is probably the weakest of the claims - it's quite easy to design a layer 2 with a general-purpose base-layer compression scheme that users can automatically extend with application-specific sub-compressors - but otherwise the use cases are all sound. But this still leaves open one large question: *is a three-layer structure the right way to accomplish these goals?* What's the point of validiums, and privacy systems, and customized environments, anchoring into layer 2 instead of just anchoring into layer 1? The answer to this question turns out to be quite complicated.
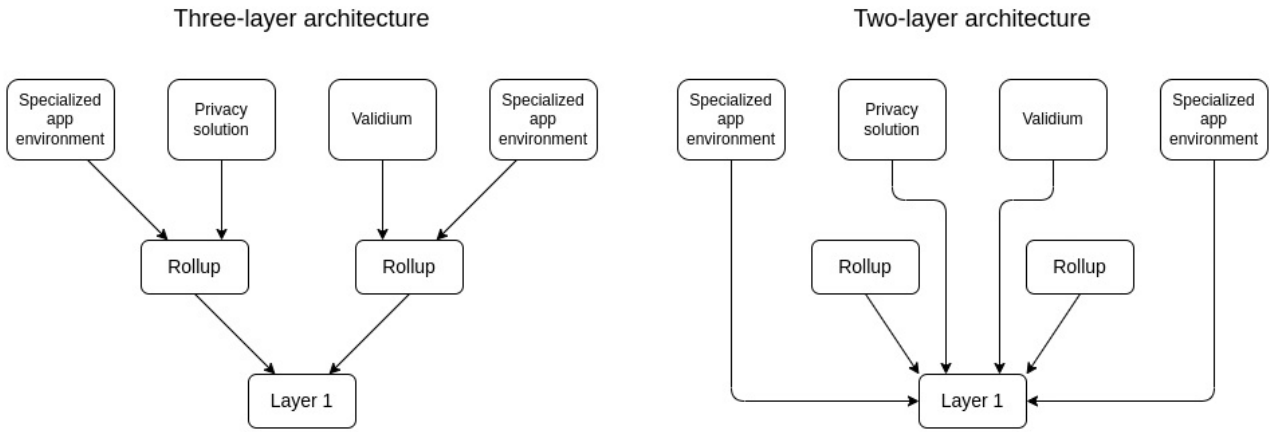
Which one is actually better?

Does depositing and withdrawing become cheaper and easier within a layer 2's sub-tree?

One possible argument for the three-layer model over the two-layer model is: a three-layer model allows an entire sub-ecosystem to exist within a single rollup, which allows cross-domain operations within that ecosystem to happen very cheaply, without needing to go through the expensive layer 1.

But as it turns out, you can do deposits and withdrawals cheaply even between two layer 2s (or even layer 3s) that commit to the same layer 1! The key realization is that **tokens and other assets do not have to be issued in the root chain**. That is, you can have an ERC20 token on Arbitrum, create a wrapper of it on Optimism, and move back and forth between the two without any L1 transactions!

Let us examine how such a system works. There are two smart contracts: the *base contract* on Arbitrum, and the *wrapper token contract* on Optimism. To move from Arbitrum to Optimism, you would send your tokens to the base contract, which would generate a receipt. Once Arbitrum finalizes, you can take a Merkle proof of that receipt, rooted in L1 state, and send it into the wrapper token contract on Optimism, which verifies it and issues you a wrapper token. To move tokens back, you do the same thing in reverse.
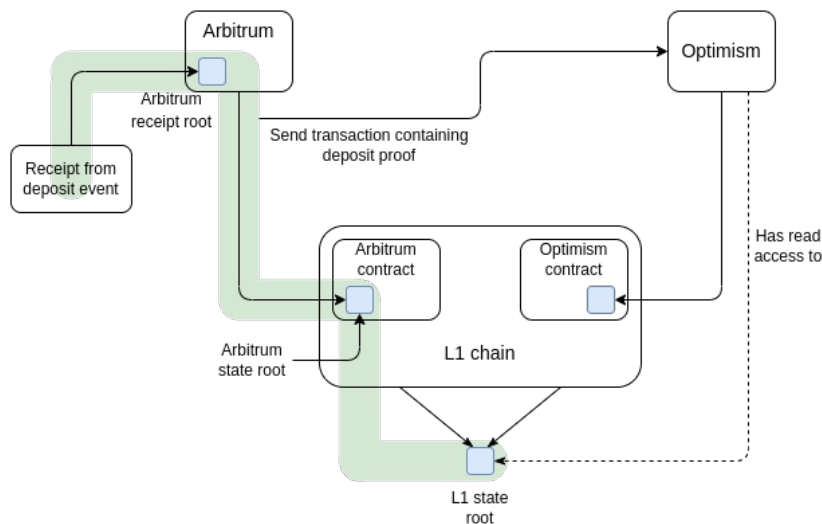
Even though the Merkle path needed to prove the deposit on Arbitrum goes through the L1 state, Optimism only needs to read the L1 state root to process the deposit - no L1 transactions required. Note that because data on rollups is the scarcest resource, a practical implementation of such a scheme would use a SNARK or a KZG proof, rather than a Merkle proof directly, to save space.

Such a scheme has one key weakness compared to tokens rooted on L1, at least on optimistic rollups: *depositing also requires waiting the fraud proof window*. If a token is rooted on L1, withdrawing from Arbitrum or Optimism back to L1 takes a week delay, but depositing is instant. In this scheme, however, both depositing and withdrawing take a week delay. That said, it's not clear that a three-layer architecture on optimistic rollups is better: there's a lot

of technical complexity in ensuring that a fraud proof game happening inside a system that itself runs on a fraud proof game is safe.

Fortunately, neither of these issues will be a problem on ZK rollups. ZK rollups do not require a week-long waiting window for security reasons, but they do still require a shorter window (perhaps 12 hours with first-generation technology) for two other reasons. First, particularly the more complex general-purpose ZK-EVM rollups need a longer amount of time to cover non-parallelizable compute time of proving a block. Second, there is the economic consideration of needing to submit proofs rarely to minimize the fixed costs associated with proof transactions. Next-gen ZK-EVM technology, including specialized hardware, will solve the first problem, and better-architected batch verification can solve the second problem. And it's precisely the issue of optimizing and batching proof submission that we will get into next.

Rollups and validiums have a confirmation time vs fixed cost tradeoff. Layer 3s can help fix this. But what else can?

The cost of a rollup *per transaction* is cheap: it's just 16-60 bytes of data, depending on the application. But rollups also have to pay a high *fixed cost* every time they submit a batch of transactions to chain: 21000 L1 gas per batch for optimistic rollups, and more than 400,000 gas for ZK rollups (millions of gas if you want something quantum-safe that only uses STARKs).

Of course, rollups could simply choose to wait until there's 10 million gas worth of L2 transactions to submit a batch, but this would give them very long batch intervals, forcing users to wait much longer until they get a high-security confirmation. Hence, they have a tradeoff: long batch intervals and optimum costs, or shorter batch intervals and greatly increased costs.

To give us some concrete numbers, let us consider a ZK rollup that has 600,000 gas per-batch costs and processes fully optimized ERC20 transfers (23 bytes), which cost 368 gas per transaction. Suppose that this rollup is in early to mid stages of adoption, and is averaging 5 TPS. We can compute gas per transaction vs batch intervals:

Batch interval	Gas per tx (= tx cost + batch cost / (TPS * batch interval))
12s (one per Ethereum block)	10368
1 min	2368
10 min	568
1 h	401

If we're entering a world with lots of customized validiums and application-specific environments, then many of them will do much less than 5 TPS. Hence, tradeoffs between confirmation time and cost start to become a very big deal. And indeed, the "layer 3" paradigm does solve this! A ZK rollup inside a ZK rollup, even implemented naively, would have fixed costs of only ~8,000 layer-1 gas (500 bytes for the proof). This changes the table above to:

Batch interval	Gas per tx (= tx cost + batch cost / (TPS * batch interval))
12s (one per Ethereum block)	501
1 min	394
10 min	370
1 h	368

Problem basically solved. So are layer 3s good? Maybe. But it's worth noticing that there is a different approach to solving this problem, inspired by ERC 4337 aggregate verification.

The strategy is as follows. Today, each ZK rollup or validium accepts a state root if it receives a proof proving that $S_{new} = STF(S_{old}, D)$: the new state root must be the result of correctly processing the transaction data or state deltas on top of the old state root. In this new scheme, the ZK rollup would accept a message from a *batch verifier contract* that says that it has verified a proof of a batch of statements, where each of those statements is of the form $S_{new} = STF(S_{old}, D)$. This batch proof could be constructed via a recursive SNARK scheme or Halo aggregation.

Proof 1:
$S_{1,new} = STF(S_{1,old}, D_1)$

Proof 2:
$S_{2,new} = STF(S_{2,old}, D_2)$

Proof 3:
$S_{3,new} = STF(S_{3,old}, D_3)$

Multi-zk-rollup batch carrier transaction

Aggregate proof

$S_{1,old}$	$S_{1,new}$	D_1
$S_{2,old}$	$S_{2,new}$	D_2
$S_{3,old}$	$S_{3,new}$	D_3

Rollup 1

$(S_{1,old}, S_{1,new}, D_1)$

Batch handler (verifies the aggregate proof)

$(S_{2,old}, S_{2,new}, D_2)$

Rollup 2

$(S_{3,old}, S_{3,new}, D_3)$
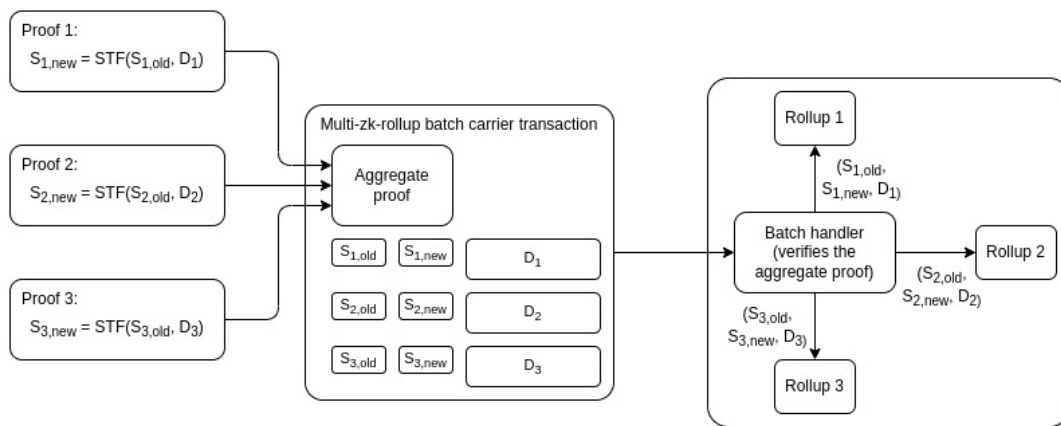
Rollup 3

This would be an open protocol: any ZK-rollup could join, and any batch prover could aggregate proofs from any compatible ZK-rollup, and would get compensated by the aggregator with a transaction fee. The batch handler contract would verify the proof once, and then pass off a message to each rollup with the (S_{old}, S_{new}, D) triple for that rollup; the fact that the triple came from the batch handler contract would be evidence that the transition is valid.

The cost per rollup in this scheme could be close to 8000 if it's well-optimized: 5000 for a state write adding the new update, 1280 for the old and new root, and an extra 1720 for miscellaneous data juggling. Hence, it would give us the same savings. Starkware actually has something like this already, called SHARP, though it is not (yet) a permissionless open protocol.

One response to this style of approach might be: *but isn't this actually just another layer 3 scheme?* Instead of `base layer <- rollup <- validium`, you have `base layer <- batch mechanism <- rollup or validium`. From some philosophical architectural standpoint, this may be true. But there is an important difference: instead of the middle layer being a complicated full EVM system, the middle layer is a simplified and highly specialized object, and so it is more likely to be secure, it is more likely to be built at all without needing yet another specialized token, and it is more likely to be governance-minimized and not change over time.

Conclusion: what even is a "layer"?

A three-layer scaling architecture that consists of stacking *the same* scaling scheme on top of itself generally does not work well. Rollups on top of rollups, where the two layers of rollups use the same technology, certainly do not. A three-layer architecture where the second layer and third layer have *different* purposes, however, can work. Validiums on top of rollups do make sense, even if they're not certain to be the long-term best way of doing things.

Once we start getting into the details of *what kind* of architecture makes sense, however, we get into the philosophical question: what is a "layer" and what is not? The `base layer <- batch mechanism <- rollup or validium` pattern does the same job as a `base layer <- rollup <- rollup or validium` pattern. But in terms of *how it works*, a proof aggregation layer looks more like ERC-4337 than like a rollup. Typically, we don't refer to ERC-4337 as a "layer 2". Similarly, we don't refer to Tornado Cash as a "layer 2" - and so if we were to be consistent, we would not refer to a privacy-focused sub-system that lives on top of a layer 2 as a layer 3. So there is an unresolved semantics debate of what deserves the title of "layer" in the first place.

There are many possible schools of thought on this. My personal preference would be to keep the term "layer 2" restricted to things with the following properties:

- Their purpose is to increase scalability
- They follow the "blockchain within a blockchain" pattern: they have their own mechanism for processing transactions and their own internal state
- They inherit the full security of the Ethereum chain

So, optimistic rollups and ZK rollups are layer 2s, but validiums, proof aggregation schemes, ERC 4337, on-chain privacy systems and Solidity are something else. It may make sense to call some of them layer 3s, but probably not all of them; in any case, it seems premature to settle definitions while the architecture of the multi-rollup ecosystem is far from set in stone and most of the discussion is happening only in theory.

That said, the language debate is less important than the technical question of which constructions actually make the most sense. There is clearly an important role to be played by "layers" of some kind that serve non-scaling needs like privacy, and there is clearly is an important function of proof aggregation that needs to be filled *somehow*, and preferably by an open protocol. But at the same time, there are good technical reasons to make the intermediary layers that connect user-facing environments to the layer 1 as simple as possible; the "glue layer" being an EVM rollup is probably not the right approach in many cases. I suspect more sophisticated (and simpler) constructions such as those described in this post will start to have a bigger role to play as the layer 2 scaling ecosystem matures.

Should there be demand-based recurring fees on ENS domains?

2022 Sep 09

Special thanks to Lars Doucet, Glen Weyl and Nick Johnson for discussion and feedback on various topics.

ENS domains today are cheap. Very cheap. The cost to register and maintain a five-letter domain name is only $5 per year. This sounds reasonable from the perspective of one person trying to register a single domain, but it looks very different when you look at the situation globally: when ENS was younger, someone could have registered all 8938 five-letter words in the Scrabble wordlist (which includes exotic stuff like "BURRS", "FLUYT" and "ZORIL") and pre-paid their ownership for a hundred years, all for the price of a dozen lambos. And in fact, many people did: today, almost all of those five-letter words are already taken, many by squatters waiting for someone to buy the domain from them at a much higher price. A random scrape of OpenSea shows that about 40% of all these domains are for sale or have been sold on that platform alone.

The question worth asking is: is this really the best way to allocate domains? By selling off these domains so cheaply, ENS DAO is almost certainly gathering far less revenue than it could, which limits its ability to act to improve the ecosystem. The status quo is also bad for fairness: being able to buy up all the domains cheaply was great for people in 2017, is okay in 2022, but the consequences may severely handicap the system in 2050. And given that buying a five-letter-word domain in practice costs anywhere from 0.1 to 500 ETH, the notionally cheap registration prices are not actually providing cost savings to users. In fact, there are deep economic reasons to believe that reliance on secondary markets makes domains *more* expensive than a well-designed in-protocol mechanism.

Could we allocate ongoing ownership of domains in a better way? Is there a way to raise more revenue for ENS DAO, do a *better* job of ensuring domains go to those who can make best use of them, and at the same time preserve the credible neutrality and the accessible very strong guarantees of long-term ownership that make ENS valuable?

Problem 1: there is a fundamental tradeoff between strength of property rights and fairness

Suppose that there are N "high-value names" (eg. five-letter words in the Scrabble dictionary, but could be any similar category). Suppose that each year, users grab up k names, and some portion p of them get grabbed by someone who's irrationally stubborn and not willing to give them up (p could be really low, it just needs to be greater than zero). Then, after $\frac{N}{k * p}$ years, no one will be able to get a high-value name again.

This is a two-line mathematical theorem, and it feels too simple to be saying anything important. But it actually gets at a crucial truth: time-unlimited allocation of a finite resource is incompatible with fairness across long time horizons. This is true for land; it's the reason why there have been so many land reforms throughout history, and it's a big part of why many advocate for land taxes today. It's also true for domains, though the problem in the traditional domain space has been temporarily alleviated by a "forced dilution" of early .com holders in the form of a mass introduction of .io, .me, .network and many other domains. ENS has soft-committed to not add new TLDs to avoid polluting the global namespace and rupturing its chances of eventual integration with mainstream DNS, so such a dilution is not an option.

Fortunately, ENS charges not just a one-time fee to register a domain, but also a recurring annual fee to maintain it. Not all decentralized domain name systems had the foresight to implement this; Unstoppable Domains did not, and even goes so far as to proudly advertise its preference for short-term consumer appeal over long-term sustainability ("No renewal fees ever!"). The recurring fees in ENS and traditional DNS are a healthy mitigation to the worst excesses of a truly unlimited pay-once-own-forever model: at the very least, the recurring fees mean that no one will be able to accidentally lock down a domain forever through forgetfulness or carelessness. But it may not be enough. It's still possible to spend $500 to lock down an ENS domain for an entire century, and there are certainly

some types of domains that are in high enough demand that this is vastly underpriced.

Problem 2: speculators do not actually create efficient markets

Once we admit that a first-come-first-serve model with low fixed fees has these problems, a common counterargument is to say: yes, many of the names will get bought up by speculators, but speculation is natural and good. It is a free market mechanism, where speculators who actually want to maximize their profit are motivated to resell the domain in such a way that it goes to whoever can make the best use of the domain, and their outsized returns are just compensation for this service.

But as it turns out, there has been academic research on this topic, and it is not actually true that profit-maximizing auctioneers maximize social welfare! Quoting [Myerson 1981](#):

> By announcing a reservation price of 50, the seller risks a probability $(1 / 2^n)$ of keeping the object even though some bidder is willing to pay more than t_0 for it; but the seller also increases his expected revenue, because he can command a higher price when the object is sold.
>
> Thus the optimal auction may not be ex-post efficient. To see more clearly why this can happen, consider the example in the above paragraph, for the case when $(n = 1)$ ... Ex post efficiency would require that the bidder must always get the object, as long as his value estimate is positive. But then the bidder would never admit to more than an infinitesimal value estimate, since any positive bid would win the object ... In fact the seller's optimal policy is to refuse to sell the object for less than 50.

Translated into diagram form:
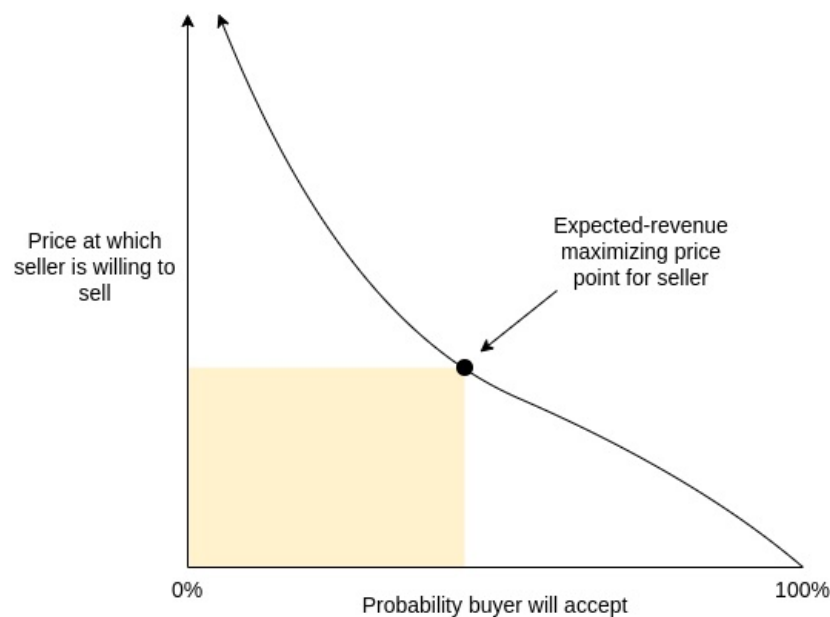
Maximizing revenue for the seller almost always requires accepting some probability of never selling the domain at all, leaving it unused outright. One important nuance in the argument is that seller-revenue-maximizing auctions are at their most inefficient when there is one possible buyer (or at least, one buyer with a valuation far above the others), and the inefficiency decreases quickly once there are many competing potential buyers. But for a large class of domains, the first category is precisely the situation they are in. Domains that are simply some person, project or company's name, for example, have one natural buyer: that person or project. And so if a speculator buys up such a name, they will of course set the price high, accepting a large chance of never coming to a deal to maximize their revenue in the case where a deal does arise.

Hence, we cannot say that speculators grabbing a large portion of domain allocation revenues is merely just compensation for them ensuring that the market is efficient. On the contrary, speculators can easily make the market *worse* than a well-designed mechanism in the protocol that encourages domains to be directly available for sale at fair prices.

One cheer for stricter property rights: stability of domain ownership has positive externalities

The monopoly problems of overly-strict property rights on non-fungible assets have been known for a long time. Resolving this issue in a market-based way was the original goal of [Harberger taxes](#): require the owner of each covered asset to set a price at which they are willing to sell it to anyone else, and charge an annual fee based on that price. For example, one could charge 0.5% of the sale price every year. Holders would be incentivized to leave the asset available for purchase at prices that are reasonable, "lazy" holders who refuse to sell would lose money every year, and hoarding assets without using them would in many cases become economically infeasible outright.

But the risk of being forced to sell something at any time can have large economic and psychological costs, and it's for this reason that advocates of Harberger taxes generally focus on industrial property applications where the market participants are sophisticated. Where do domains fall on the spectrum? Let us consider the costs of a business getting "relocated", in three separate cases: a data center, a restaurant, and an ENS name.

	Data center	Restaurant	ENS name
Confusion from people expecting old location	An employee comes to the old location, and unexpectedly finds it closed.	An employee or a customer comes to the old location, and unexpectedly finds it closed.	Someone sends a big chunk of money to the wrong address.
Loss of location-specific long-term investment	Low	The restaurant will probably lose many long-term customers for whom the new location is too far away	The owner spent years building a brand around the old name that cannot easily carry over.

As it turns out, domains do not hold up very well. Domain name owners are often not sophisticated, the costs of switching domain names are often high, and negative externalities of a name-change gone wrong can be large. The highest-value owner of `coinbase.eth` may not be Coinbase; it could just as easily be a scammer who would grab up the domain and then immediately make a fake charity or ICO claiming it's run by Coinbase and ask people to send that address their money. **For these reasons, Harberger taxing domains is not a great idea**.

Alternative solution 1: demand-based recurring pricing

Maintaining ownership over an ENS domain today requires paying a recurring fee. For most domains, this is a simple and very low $5 per year. The only exceptions are four-letter domains ($160 per year) and three-letter domains ($640 per year). But what if instead, we make the fee somehow depend on the actual level of market demand for the domain?

This would not be a Harberger-like scheme where you have to make the domain *available for immediate sale* at a particular price. Rather, the initiative in the price-setting procedure would fall on the bidders. Anyone could bid on a particular domain, and if they keep an open bid for a sufficiently long period of time (eg. 4 weeks), the domain's valuation rises to that level. The annual fee on the domain would be proportional to the valuation (eg. it might be set to 0.5% of the valuation). If there are no bids, the fee might decay at a constant rate.
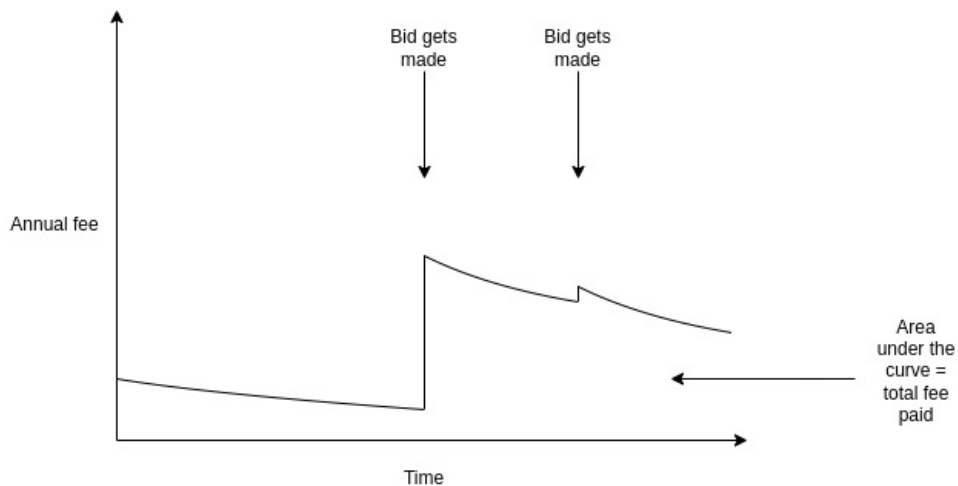
When a bidder sends their bid amount into a smart contract to place a bid, the owner has two options: they could either accept the bid, or they could reject, though they may have to start paying a higher price. If a bidder bids a value higher than the actual value of the domain, the owner could sell to them, costing the bidder a huge amount of money.

This property is important, because it means that "griefing" domain holders is risky and expensive, and may even end up benefiting the victim. If you own a domain, and a powerful actor wants to harass or censor you, they could try to make a very high bid for that domain to greatly increase your annual fee. But if they do this, you could simply sell to them and collect the massive payout.

This already provides much more stability and is more noob-friendly than a Harberger tax. Domain owners don't need to constantly worry whether or not they're setting prices too low. Rather, they can simply sit back and pay the annual fee, and if someone offers to bid they can take 4 weeks to make a decision and either sell the domain or continue holding it and accept the higher fee. But even this probably does not provide quite enough stability. To go even further, we need a compromise on the compromise.

Alternative solution 2: capped demand-based recurring pricing

We can modify the above scheme to offer even stronger guarantees to domain-name holders. Specifically, we can try to offer the following property:

> **Strong time-bound ownership guarantee**: for any fixed number of years, it's always possible to compute a fixed amount of money that you can pre-pay to unconditionally guarantee ownership for *at least* that number of years.

In math language, there must be some function $y = f(n)$ such that if you pay y dollars (or ETH), you get a hard guarantee that you will be able to hold on to the domain for at least n years, no matter what happens. f *may* also depend on other factors, such as what happened to the domain previously, as long as those factors are known at the time the transaction to register or extend a domain is made. Note that the *maximum annual fee* after n years would be the derivative $f'(n)$.

The new price after a bid would be capped at the implied maximum annual fee. For example, if $f(n) = \frac{1}{2}n^2$, so $f'(n) = n$, and you get a bid of \$5 after 7 years, the annual fee would rise to \$5, but if you get a bid of \$10 after 7 years, the annual fee would only rise to \$7. If no bids that raise the fee to the max are made for some length of time (eg. a full year), n resets. If a bid is made and rejected, n resets.

And of course, we have a highly subjective criterion that $f(n)$ must be "reasonable". We can create compromise proposals by trying different shapes for f:

$f(n)$ (p_0 = price of last sale)		Total cost to	Total cost to

Type	or last rejected bid, or $1 if most recent event is a reset)	In plain English	guarantee holding for >= 10 years	guarantee holding for >= 100 years
Exponential fee growth	$f(n) = \int_0^n p_0 * 1.1^n$	The fee can grow by a maximum of 10% per year (with compounding).	$836	$7.22m
Linear fee growth	$f(n) = p_0 * n + \frac{15}{2}n^2$	The annual fee can grow by a maximum of $15 per year.	$1250	$80k
Capped annual fee	$f(n) = 640 * n$	The annual fee cannot exceed $640 per year. That is, a domain in high demand can start to cost as much as a three-letter domain, but not more.	$6400	$64k

Or in chart form:
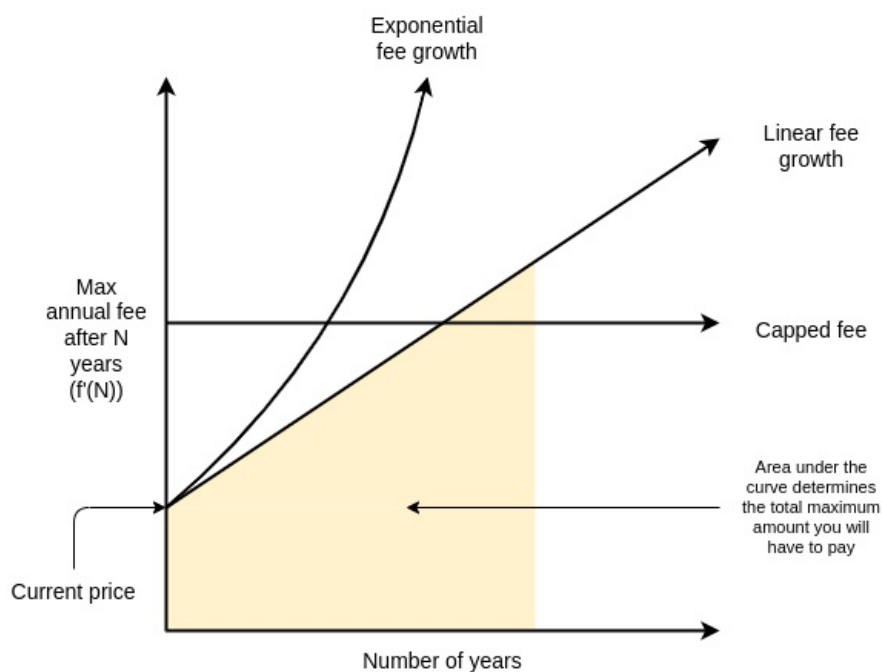
Note that the amounts in the table are only the *theoretical maximums needed to guarantee* holding a domain for that number of years. In practice, almost no domains would have bidders willing to bid very high amounts, and so holders of almost all domains would end up paying much less than the maximum.

One fascinating property of the "capped annual fee" approach is that there are versions of it that are strictly *more favorable* to existing domain-name holders than the status quo. In particular, we could imagine a system where a domain that gets no bids does not have to pay *any* annual fee, and a bid could increase the annual fee to a maximum of $5 per year.

Demand from external bids clearly provides *some* signal about how valuable a domain is (and therefore, to what extent an owner is excluding others by maintaining control over it). Hence,

regardless of your views on what level of fees should be required to maintain a domain, I would argue that you should find *some* parameter choice for demand-based fees appealing.

I will still make my case for why some *superlinear* $f(n)$, a max annual fee that goes up over time, is a good idea. First, paying more for longer-term security is a common feature throughout the economy. Fixed-rate mortgages usually have [higher interest rates](#) than variable-rate mortgages. You can *get* higher interest by providing deposits that are locked up for longer periods of time; this is compensation the bank pays *you* for providing longer-term security *to the bank*. Similarly, longer-term government bonds typically have [higher yields](#). Second, the annual fee should be able to *eventually* adjust to whatever the market value of the domain is; we just don't want that to happen too quickly.

Superlinear $f(n)$ values still make hard guarantees of ownership reasonably accessible over pretty long timescales: with the linear-fee-growth formula $f(n) = p\_0 * n + \frac{15}{2}n^2$, for only $6000 ($120 per year) you could ensure ownership of the domain for 25 years, and you would almost certainly pay much less. The ideal of "register and forget" for censorship-resistant services would still be very much available.

From here to there

Weakening property norms, and increasing fees, is psychologically very unappealing to many people. This is true even when these fees make clear economic sense, and even when you can redirect fee revenue into a UBI and mathematically show that the majority of people would economically net-benefit from your proposal. Cities have a hard time [adding congestion pricing](#), even when it's painfully clear that the only two choices are paying congestion fees in dollars and paying congestion fees in wasted time [and weakened mental health](#) driving in painfully slow traffic. [Land value taxes](#), despite being in many ways [one of the most effective and least harmful taxes](#) out there, have a hard time getting adopted. Unstoppable Domains's loud and proud advertisement of "no renewal fees ever" is in my view very short-sighted, but it's clearly at least somewhat *effective*. So how could I possibly think that we have any chance of adding fees and conditions to domain name ownership?

The crypto space is not going to solve deep challenges in human political psychology that humanity has failed at [for centuries](#). But we do not have to. I see two possible answers that *do* have some realistic hope for success:

1. **Democratic legitimacy: come up with a compromise proposal that really is a sufficient compromise that it makes enough people happy**, and perhaps even makes some existing domain name holders (not just *potential* domain name holders) *better off* than they are today.

   For example, we could implement demand-based annual fees (eg. setting the annual fee to 0.5% of the highest bid) with a fee cap of $640 per year for domains up to eight letters long, and $5 per year for longer domains, and let domain holders pay nothing if no one makes a bid. Many average users would *save* money under such a proposal.

2. **Market legitimacy: avoid the need to get legitimacy to overturn people's expectations in the existing system by instead creating a new system** (or sub-system).

   In traditional DNS, this could be done just by creating a new TLD that would be as convenient as existing TLDs. In ENS, there is a stated desire to stick to `.eth` only to avoid conflicting with the existing domain name system. And using existing subdomains doesn't quite work: `foo.bar.eth` is much less nice than `foo.eth`. One possible middle route is for the ENS DAO to hand off *single-letter* domain names solely to projects that run some other kind of credibly-neutral marketplace for their subdomains, as long as they hand over at least 50% of the revenue to the ENS DAO.

   For example, perhaps `x.eth` could use one of my proposed pricing schemes for its subdomains, and `t.eth` could implement a mechanism where ENS DAO has the right to forcibly transfer subdomains for anti-fraud and trademark reasons. `foo.x.eth` *just barely* looks good enough to be sort-of a substitute for `foo.eth`; it will have to do.

If making changes to ENS domain pricing itself are off the table, then the market-based approach of explicitly encouraging marketplaces with different rules in subdomains should be strongly considered.

To me, the crypto space is not just about coins, and I admit my attraction to ENS does not center around some notion of *unconditional and infinitely strict* property-like ownership over domains. Rather, my interest in the space lies more in [credible neutrality](#), and property rights that are strongly

protected particularly against politically motivated censorship and arbitrary and targeted interference by powerful actors. That said, a high degree of guarantee of ownership is nevertheless very important for a domain name system to be able to function.

The hybrid proposals I suggest above are my attempt at preserving total credible neutrality, continuing to provide a high degree of ownership guarantee, but at the same time increasing the cost of domain squatting, raising more revenue for the ENS DAO to be able to work on important public goods, and *improving* the chances that people who do not have the domain they want already will be able to get one.

The different types of ZK-EVMs

2022 Aug 04

Special thanks to the PSE, Polygon Hermez, Zksync, Scroll, Matter Labs and Starkware teams for discussion and review.

There have been many "ZK-EVM" projects making flashy announcements recently. [Polygon](#) open-sourced their ZK-EVM project, [ZKSync](#) released their plans for ZKSync 2.0, and the relative newcomer [Scroll](#) announced their ZK-EVM recently. There is also the ongoing effort from the [Privacy and Scaling Explorations team](#), [Nicolas Liochon et al's team](#), an [alpha compiler](#) from the EVM to Starkware's ZK-friendly language [Cairo](#), and certainly at least a few others I have missed.

The core goal of all of these projects is the same: to use [ZK-SNARK](#) technology to make cryptographic proofs of execution of Ethereum-like transactions, either to make it much easier to verify the Ethereum chain itself or to build [ZK-rollups](#) that are (close to) equivalent to what Ethereum provides but are much more scalable. But there are subtle differences between these projects, and what tradeoffs they are making between practicality and speed. This post will attempt to describe a taxonomy of different "types" of EVM equivalence, and what are the benefits and costs of trying to achieve each type.

Overview (in chart form)
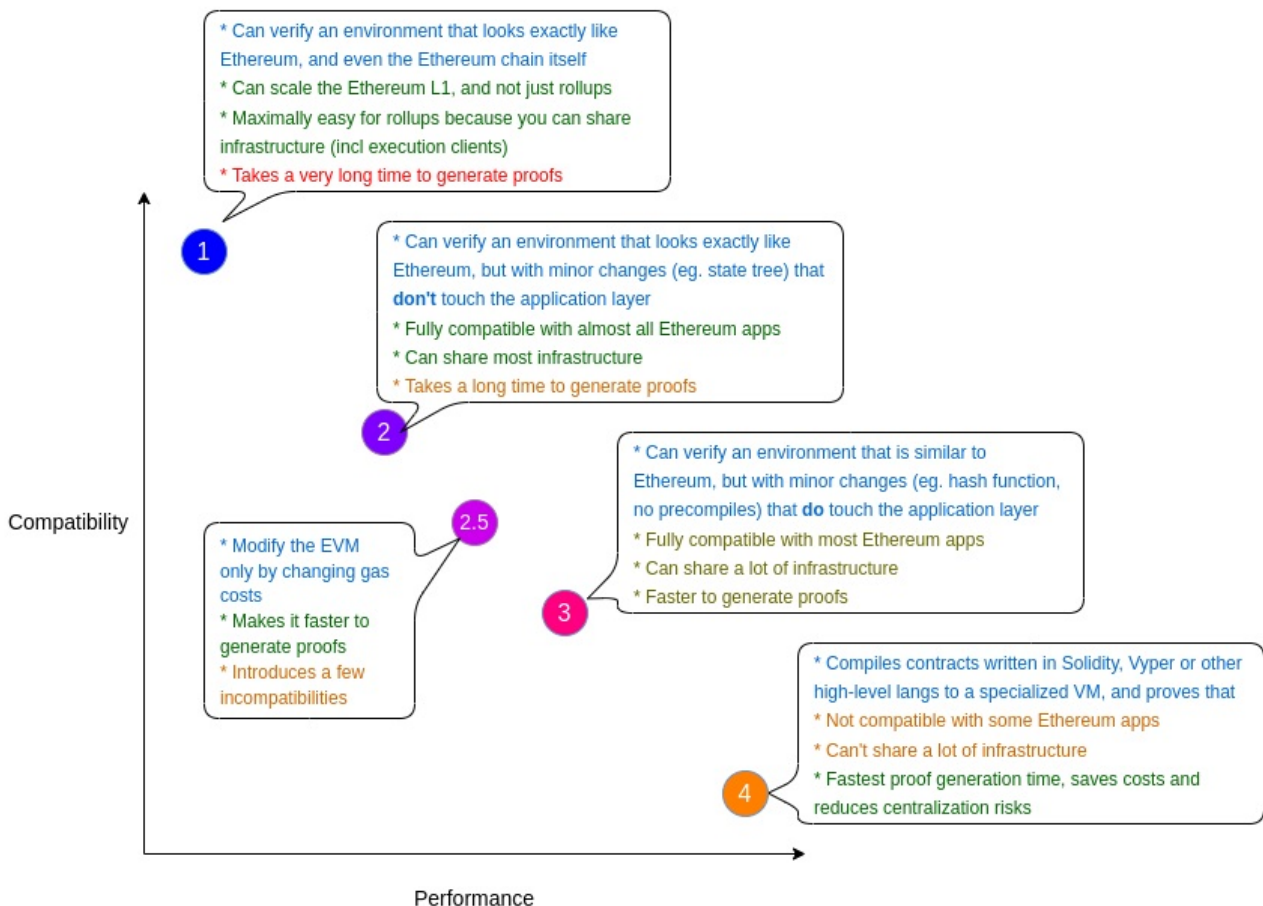
Type 1 (fully Ethereum-equivalent)

Type 1 ZK-EVMs strive to be fully and uncompromisingly Ethereum-equivalent. They do not change any part of the Ethereum system to make it easier to generate proofs. They do not replace hashes, state trees, transaction trees, precompiles or any other in-consensus logic, no matter how peripheral.

Advantage: perfect compatibility

The goal is to be able to verify Ethereum blocks as they are today - or at least, verify the execution-layer side (so, beacon chain consensus logic is not included, but all the transaction execution and smart contract and account logic is included).

Type 1 ZK-EVMs are what we ultimately need make the Ethereum layer 1 itself more scalable. In the long term, modifications to Ethereum tested out in Type 2 or Type 3 ZK-EVMs might be introduced into Ethereum proper, but such a re-architecting comes with its own complexities.

Type 1 ZK-EVMs are also ideal for rollups, because they allow rollups to re-use a lot of infrastructure. For example, Ethereum execution clients can be used as-is to generate and process rollup blocks (or at least, they can be once withdrawals are implemented and that functionality can be re-used to support ETH being deposited into the rollup), so tooling such as block explorers, block production, etc is very easy to re-use.

Disadvantage: prover time

Ethereum was not originally designed around ZK-friendliness, so there are *many* parts of the Ethereum protocol that take a large amount of computation to ZK-prove. Type 1 aims to replicate Ethereum exactly, and so it has no way of mitigating these inefficiencies. At present, proofs for Ethereum blocks take many hours to produce. This can be mitigated either by clever engineering to massively parallelize the prover or in the longer term by ZK-SNARK ASICs.

Who's building it?

The ZK-EVM Community Edition (bootstrapped by community contributors including Privacy and Scaling Explorations, the Scroll team, Taiko and others) is a Tier 1 ZK-EVM.

Type 2 (fully EVM-equivalent)

Type 2 ZK-EVMs strive to be exactly EVM-equivalent, but not quite Ethereum-equivalent. That is, they look exactly like Ethereum "from within", but they have some differences on the outside, particularly in data structures like the block structure and state tree.

The goal is to be fully compatible with existing applications, but make some minor modifications to Ethereum to make development easier and to make proof generation faster.

Advantage: perfect equivalence at the VM level

Type 2 ZK-EVMs make changes to data structures that hold things like the Ethereum state. Fortunately, these are structures that the EVM itself cannot access directly, and so applications that work on Ethereum would almost always still work on a Type 2 ZK-EVM rollup. You would not be able to use Ethereum execution clients as-is, but you could use them with some modifications, and you would still be able to use EVM debugging tools and most other developer infrastructure.

There are a small number of exceptions. One incompatibility arises for applications that verify Merkle proofs of historical Ethereum blocks to verify claims about historical transactions, receipts or state (eg. bridges sometimes do this). A ZK-EVM that replaces Keccak with a different hash function would break these proofs. However, I usually recommend against building applications this way anyway, because future Ethereum changes (eg. Verkle trees) will break such applications even on Ethereum itself. A better alternative would be for Ethereum itself to add future-proof history access precompiles.

Disadvantage: improved but still slow prover time

Type 2 ZK-EVMs provide faster prover times than Type 1 mainly by removing parts of the Ethereum stack that rely on needlessly complicated and ZK-unfriendly cryptography. Particularly, they might change Ethereum's Keccak and RLP-based Merkle Patricia tree and perhaps the block and receipt structures. Type 2 ZK-EVMs might instead use a different hash function, eg. Poseidon. Another natural modification is modifying the state tree to store the code hash and keccak, removing the need to verify hashes to process the `EXTCODEHASH` and `EXTCODECOPY` opcodes.

These modifications significantly improve prover times, but they do not solve every problem. The slowness from having to prove the EVM as-is, with all of the inefficiencies and ZK-unfriendliness inherent to the EVM, still remains. One simple example of this is memory: because an `MLOAD` can read any 32 bytes, including "unaligned" chunks (where the start and end are not multiples of 32), an MLOAD can't simply be interpreted as reading one chunk; rather, it might require reading two

consecutive chunks and performing bit operations to combine the result.

Scroll's ZK-EVM project is building toward a Type 2 ZK-EVM, as is Polygon Hermez. That said, neither project is quite there yet; in particular, a lot of the more complicated precompiles have not yet been implemented. Hence, at the moment both projects are better considered Type 3.

Type 2.5 (EVM-equivalent, except for gas costs)

One way to significantly improve worst-case prover times is to greatly increase the gas costs of specific operations in the EVM that are very difficult to ZK-prove. This might involve precompiles, the KECCAK opcode, and possibly specific patterns of calling contracts or accessing memory or storage or reverting.

Changing gas costs may reduce developer tooling compatibility and break a few applications, but it's generally considered less risky than "deeper" EVM changes. Developers should take care to not require more gas in a transaction than fits into a block, to never make calls with hard-coded amounts of gas (this has already been standard advice for developers for a long time).

An alternative way to manage resource constraints is to simply set hard limits on the number of times each operation can be called. This is easier to implement in circuits, but plays much less nicely with EVM security assumptions. I would call this approach Type 3 rather than Type 2.5.

Type 3 (almost EVM-equivalent)

Type 3 ZK-EVMs are *almost* EVM-equivalent, but make a few sacrifices to exact equivalence to further improve prover times and make the EVM easier to develop.

Advantage: easier to build, and faster prover times

Type 3 ZK-EVMs might remove a few features that are exceptionally hard to implement in a ZK-EVM implementation. Precompiles are often at the top of the list here;. Additionally, Type 3 ZK-EVMs sometimes also have minor differences in how they treat contract code, memory or stack.

Disadvantage: more incompatibility

The goal of a Type 3 ZK-EVM is to be compatible with *most* applications, and require only minimal re-writing for the rest. That said, there will be some applications that would need to be rewritten either because they use pre-compiles that the Type 3 ZK-EVM removes or because of subtle dependencies on edge cases that the VMs treat differently.

Scroll and Polygon are both Type 3 in their current forms, though they're expected to improve compatibility over time. Polygon has a unique design where they are ZK-verifying their own internal language called zkASM, and they interpret ZK-EVM code using the zkASM implementation. Despite this implementation detail, I would still call this a genuine Type 3 ZK-EVM; it can still verify EVM code, it just uses some different internal logic to do it.

Today, no ZK-EVM team *wants* to be a Type 3; Type 3 is simply a transitional stage until the complicated work of adding precompiles is finished and the project can move to Type 2.5. In the future, however, Type 1 or Type 2 ZK-EVMs may become Type 3 ZK-EVMs voluntarily, by adding in *new* ZK-SNARK-friendly precompiles that provide functionality for developers with low prover times and gas costs.

Type 4 (high-level-language equivalent)

A Type 4 system works by taking smart contract source code written in a high-level language (eg. Solidity, Vyper, or some intermediate that both compile to) and compiling *that* to some language that is explicitly designed to be ZK-SNARK-friendly.

Advantage: very fast prover times

There is a *lot* of overhead that you can avoid by not ZK-proving all the different parts of each EVM execution step, and starting from the higher-level code directly.

I'm only describing this advantage with one sentence in this post (compared to a big bullet point list below for compatibility-related disadvantages), but that should not be interpreted as a value judgement! Compiling from high-level languages directly really can greatly reduce costs and help decentralization by making it easier to be a prover.

Disadvantage: more incompatibility

A "normal" application written in Vyper or Solidity can be compiled down and it would "just work", but there are some important ways in which very many applications are not "normal":

- **Contracts may not have the same addresses** in a Type 4 system as they do in the EVM, because CREATE2 contract addresses depend on the exact bytecode. This breaks applications that rely on not-yet-deployed "counterfactual contracts", ERC-4337 wallets, [EIP-2470 singletons](#) and many other applications.
- **Handwritten EVM bytecode** is more difficult to use. Many applications use handwritten EVM bytecode in some parts for efficiency. Type 4 systems may not support it, though there are ways to implement limited EVM bytecode support to satisfy these use cases without going through the effort of becoming a full-on Type 3 ZK-EVM.
- **Lots of debugging infrastructure cannot be carried over**, because such infrastructure runs over the EVM bytecode. That said, this disadvantage is mitigated by the *greater* access to debugging infrastructure from "traditional" high-level or intermediate languages (eg. LLVM).

Developers should be mindful of these issues.

Who's building it?

[ZKSync](#) is a Type 4 system, though it may add compatibility for EVM bytecode over time. Nethermind's [Warp](#) project is building a compiler from Solidity to Starkware's Cairo, which will turn StarkNet into a de-facto Type 4 system.

The future of ZK-EVM types

The types are not unambiguously "better" or "worse" than other types. Rather, they are different points on the tradeoff space: lower-numbered types are more compatible with existing infrastructure but slower, and higher-numbered types are less compatible with existing infrastructure but faster. In general, it's healthy for the space that all of these types are being explored.

Additionally, ZK-EVM projects can easily start at higher-numbered types and jump to lower-numbered types (or vice versa) over time. For example:

- A ZK-EVM could start as Type 3, deciding not to include some features that are especially hard to ZK-prove. Later, they can add those features over time, and move to Type 2.
- A ZK-EVM could start as Type 2, and later become a hybrid Type 2 / Type 1 ZK-EVM, by providing the possibility of operating either in full Ethereum compatibility mode or with a modified state tree that can be proven faster. Scroll is considering moving in this direction.
- What starts off as a Type 4 system could become Type 3 over time by adding the ability to process EVM code later on (though developers would still be encouraged to compile direct from high-level languages to reduce fees and prover times)
- A Type 2 or Type 3 ZK-EVM can become a Type 1 ZK-EVM if Ethereum itself adopts its modifications in an effort to become more ZK-friendly.
- A Type 1 or Type 2 ZK-EVM can become a Type 3 ZK-EVM by adding a precompile for verifying code in a very ZK-SNARK-friendly language. This would give developers a choice between Ethereum compatibility and speed. This would be Type 3, because it breaks perfect EVM equivalence, but for practical intents and purposes it would have a lot of the benefits of Type 1 and 2. The main downside might be that some developer tooling would not understand the ZK-EVM's custom precompiles, though this could be fixed: developer tools could add universal precompile support by supporting a config format that includes an EVM code equivalent implementation of the precompile.

Personally, my hope is that everything becomes Type 1 over time, through a combination of improvements in ZK-EVMs and improvements to Ethereum itself to make it more ZK-SNARK-friendly. In such a future, we would have multiple ZK-EVM implementations which could be used both for ZK rollups and to verify the Ethereum chain itself. Theoretically, there is no need for Ethereum to standardize on a single ZK-EVM implementation for L1 use; different clients could use different proofs, so we continue to benefit from code redundancy.

However, it is going to take quite some time until we get to such a future. In the meantime, we are going to see a lot of innovation in the different paths to scaling Ethereum and Ethereum-based ZK-

rollups.

What do I think about network states?

2022 Jul 13

On July 4, Balaji Srinivasan released [the first version of his long-awaited new book](#) describing his vision for "**network states**": communities organized around a particular vision of how to run their own society that start off as online clubs, but then build up more and more of a presence over time and eventually become large enough to seek political autonomy or even diplomatic recognition.

Network states can be viewed as an attempt at an ideological successor to libertarianism: Balaji repeatedly praises [The Sovereign Individual](#) (see my mini-review [here](#)) as important reading and inspiration, but also departs from its thinking in key ways, centering in his new work many non-individualistic and non-monetary aspects of social relations like morals and community. Network states can also be viewed as an attempt to sketch out a possible broader political narrative for the crypto space. Rather than staying in their own corner of the internet disconnected from the wider world, blockchains could serve as a centerpiece for a new way of organizing large chunks of human society.

These are high promises. Can network states live up to them? Do network states actually provide enough benefits to be worth getting excited about? Regardless of the merits of network states, does it actually make sense to tie the idea together with blockchains and cryptocurrency? And on the other hand, is there anything crucially important that this vision of the world misses? This post represents my attempt to try to understand these questions.

Table of contents

What is a network state?

Balaji helpfully gives multiple short definitions of what a network state is. First, his definition in one sentence:

> A network state is a highly aligned online community with a capacity for collective action that crowdfunds territory around the world and eventually gains diplomatic recognition from pre-existing states.

This so far seems uncontroversial. Create a new internet community online, once it grows big enough materialize it offline, and eventually try to negotiate for some kind of status. Someone of almost *any* political ideology could find *some* form of network state under this definition that they could get behind. But now, we get to his definition in a *longer* sentence:

> A network state is a social network with a moral innovation, a sense of national consciousness, a recognized founder, a capacity for collective action, an in-person level of civility, an integrated cryptocurrency, a consensual government limited by a social smart contract, an archipelago of crowdfunded physical territories, a virtual capital, and an on-chain census that proves a large enough population, income, and real-estate footprint to attain a measure of diplomatic recognition.

Here, the concept starts to get opinionated: we're not just talking about the general concept of online communities that have collective agency and eventually try to materialize on land, we're talking about a *specific Balajian vision* of what network states should look like. It's completely possible to support network states in general, but have disagreements with the Balajian view of what properties

network states should have. If you're not already a "crypto convert", it's hard to see why an "integrated cryptocurrency" is such a fundamental part of the network state concept, for example - though Balaji does later on in the book defend his choices.

Finally, Balaji expands on this conception of a Balajian network state in longer-form, first in "a thousand words" (apparently, Balajian network states use base 8, as the actual word count is exactly \(512 = 8^3\)) and then an essay, and at the very end of the book a whole chapter.
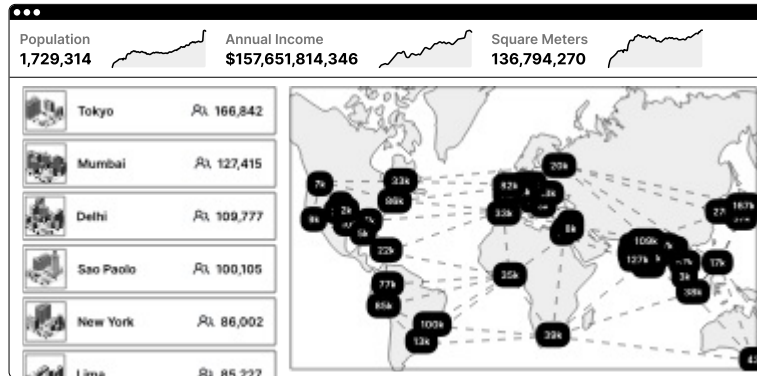
And, of course, an image.

One key point that Balaji stresses across many chapters and pages is the unavoidable *moral* ingredient required for any successful new community. As Balaji writes:

> The quick answer comes from Paul Johnson at the 11:00 mark of this talk, where he notes that early America's religious colonies succeeded at a higher rate than its for-profit colonies, because the former had a purpose. The slightly longer answer is that in a startup society, you're not asking people to buy a product (which is an economic, individualistic pitch) but to join a community (which is a cultural, collective pitch).

The commitment paradox of religious communes is key here: counterintuitively, it's the religious communes that *demand the most* of their members that are the most long-lasting.
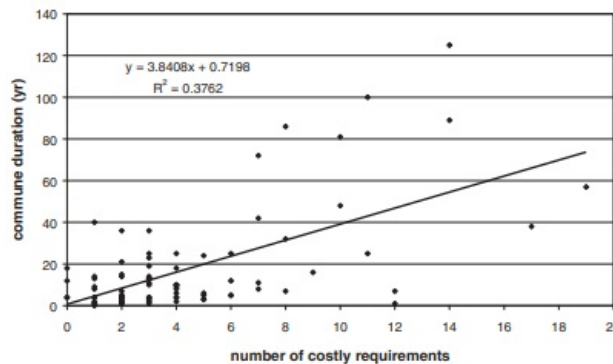
This is where Balajism explicitly diverges from the more traditional neoliberal-capitalist ideal of the defanged, apolitical and passion-free consumerist "last man". Unlike the strawman libertarian, Balaji does not believe that everything can "merely be a consumer product". Rather, he stresses greatly the importance of social norms for cohesion, and a literally religious attachment to the values that make a particular network state distinct from the world outside. As Balaji says in this podcast at 18:20, most current libertarian attempts at micronations are like "Zionism without Judaism", and this is a key part of why they fail.

This recognition is not a new one. Indeed, it's at the core of Antonio Garcia Martinez's criticism of Balaji's earlier sovereign-individual ideas (see this podcast at ~27:00), praising the tenacity of Cuban exiles in Miami who "perhaps irrationally, said this is our new homeland, this is our last stand". And

in Fukuyama's *The End of History*:

> This city, like any city, has foreign enemies and needs to be defended from outside attack. It therefore needs a class of guardians who are courageous and public-spirited, who are willing to sacrifice their material desires and wants for the sake of the common good. Socrates does not believe that courage and public-spiritedness can arise out of a calculation of enlightened self-interest. Rather, they must be rooted in thymos, in the just pride of the guardian class in themselves and in their own city, and their potentially irrational anger against those who threaten it.

Balaji's argument in *The Network State*, as I am interpreting it, is as follows. While we do need political collectives bound not just by economic interest but also by moral force, we don't need to stick with *the specific political collectives we have today*, which are highly flawed and increasingly unrepresentative of people's values. Rather, we can, and should, create new and better collectives - and his [seven-step program](#) tells us how.

So what kinds of network states could we build?

Balaji outlines a few ideas for network states, which I will condense into two key directions: **lifestyle immersion** and **pro-tech regulatory innovation**.

Balaji's go-to example for lifestyle immersion is a network state organized around health:

> Next, let's do an example which requires a network archipelago (with a physical footprint) but not a full network state (with diplomatic recognition). This is **Keto Kosher, the sugar-free society**.

> Start with a history of the horrible USDA Food Pyramid, the grain-heavy monstrosity that gave cover to the corporate sugarification of the globe and the obesity epidemic. ... Organize a community online that crowdfunds properties around the world, like apartment buildings and gyms, and perhaps eventually even culdesacs and small towns. You might take an extreme sugar teetotaler approach, literally banning processed foods and sugar at the border, thereby implementing a kind of "Keto Kosher".

> You can imagine variants of this startup society that are like "Carnivory Communities" or "Paleo People". These would be competing startup societies in the same broad area, iterations on a theme. If successful, such a society might not stop at sugar. It could get into setting cultural defaults for fitness and exercise. Or perhaps it could bulk purchase continuous glucose meters for all members, or orders of metformin.

This, strictly speaking, does not require any diplomatic recognition or even political autonomy - though perhaps, in the longer-term future, such enclaves could negotiate for lower health insurance fees and medicare taxes for their members. What does require autonomy? Well, how about a free zone for medical innovation?

> Now let's do a more difficult example, which will require a full network state with diplomatic recognition. This is **the medical sovereignty zone, the FDA-free society**.

> You begin your startup society with Henninger's history of FDA-caused [drug lag](#) and Tabarrok's history of FDA interference with so-called ["off label" prescription](#). You point out how many millions were killed by its policies, hand out t-shirts like ACT-UP did, show Dallas Buyers Club to all prospective residents, and make clear to all new members why your cause of medical sovereignty is righteous ...

> For the case of doing it outside the US, your startup society would ride behind, say, the support of the Malta's FDA for a new biomedical regime. For the case of doing it within the US, you'd need a governor who'd declare a sanctuary state for biomedicine. That is, just like a sanctuary city declares that it won't enforce federal immigration law, a sanctuary state for biomedicine would not enforce FDA writ.

One can think up of many more examples for both categories. One could have a zone where it's okay to walk around naked, both securing your legal right to do so and helping you *feel comfortable* by creating an environment where many other people are naked too. Alternatively, you could have a zone where everyone can only wear basic plain-colored clothing, to discourage what's perceived as a zero-sum status competition of expending huge effort to look better than everyone else. One could have an intentional community zone for cryptocurrency users, requiring every store to accept it and demanding an NFT to get in the zone at all. Or one could build an enclave that legalizes radical

experiments in transit and drone delivery, accepting higher risks to personal safety in exchange for the privilege of participating in a technological frontier that will hopefully set examples for the world as a whole.

What is common about all of these examples is the value of having a physical region, at least of a few hectares, where the network state's unique rules are enforced. Sure, you could individually insist on only eating at healthy restaurants, and research each restaurant carefully before you go there. But it's just *so much easier* to have a defined plot of land where you have an assurance that anywhere you go within that plot of land will meet your standards. Of course, you could lobby your local government to tighten health and safety regulations. But if you do that, you risk friction with people who have radically different preferences on tradeoffs, and you risk [shutting poor people out of the economy](). A network state offers a moderate approach.

What is Balaji's megapolitical case for network states?

One of the curious features of the book that a reader will notice almost immediately is that it sometimes feels like two books in one: sometimes, it's a book about the concept of network states, and at other times it's an exposition of Balaji's grand megapolitical theory.

Balaji's grand megapolitical theory is pretty out-there and fun in a bunch of ways. Near the beginning of the book, he entices readers with tidbits like... ok fine, I'll just quote:

- Germany sent [Vladimir Lenin]() into Russia, potentially as part of a [strategy]() to destabilize their then-rival in war. Antony Sutton's books document how some [Wall Street]() bankers apparently funded the Russian Revolution (and how other Wall Street bankers [funded the Nazis]() years later). Leon Trotsky spent [time in New York]() prior to the revolution, and propagandistic reporting from Americans like [John Reed]() aided Lenin and Trotsky in their revolution. Indeed, Reed was so useful to the Soviets — and so misleading as to the nature of the revolution — that he was [buried at the base of the Kremlin Wall](). Surprise: the Russian Revolution wasn't done wholly by Russians, but had significant foreign involvement from Germans and Americans.
- The Ochs-Sulzberger family, which owns The New York Times Company, [owned slaves]() but didn't report that fact in their [1619 coverage]().
- New York Times correspondent [Walter Duranty]() won a [Pulitzer Prize]() for helping the Soviet Union starve Ukraine into submission, 90 years before the Times decided to instead "[stand with Ukraine]()".

You can find a bunch more juicy examples in the chapter titled, appropriately, "[If the News is Fake, Imagine History]()". These examples seem haphazard, and indeed, to some extent they are so intentionally: the goal is first and foremost to shock the reader out of their existing world model so they can start downloading Balaji's own.

But pretty soon, Balaji's examples do start to point to some particular themes: a deep dislike of the "woke" US left, exemplified by the New York Times, a combination of strong discomfort with the Chinese Communist Party's authoritarianism with an understanding of why the CCP often justifiably fears the United States, and an appreciation of the love of freedom of the US right (exemplified by Bitcoin maximalists) combined with a dislike of their hostility toward cooperation and order.

Next, we get Balaji's overview of [the political realignments in recent history](), and finally we get to his core model of politics in the present day: [NYT, CCP, BTC]().

Team NYT basically runs the US, and its total lack of competence means that the US is collapsing. Team BTC (meaning, both actual Bitcoin maximalists and US rightists in general) has some positive values, but their outright hostility to collective action and order means that they are incapable of building anything. Team CCP can build, but they are building a dystopian surveillance state that much of the world would not want to live in. And all three teams are waaay too nationalist: they view things from the perspective of their own country, and ignore or exploit everyone else. Even when the teams are internationalist in theory, their specific ways of interpreting their values make them unpalatable outside of a small part of the world.

Network states, in Balaji's view, are a "de-centralized center" that could create a better alternative. They combine the love of freedom of team BTC with the moral energy of team NYT and the organization of team CCP, and give us the best benefits of all three (plus a level of international appeal greater than *any* of the three) and avoid the worst parts.

This is Balajian megapolitics in a nutshell. It is not trying to justify network states using some abstract theory (eg. some Dunbar's number or concentrated-incentive argument that the optimal size of a political body is actually in the low tens of thousands). Rather, it is an argument that situates network states as a response to the particular political situation of the world at its current place and time.
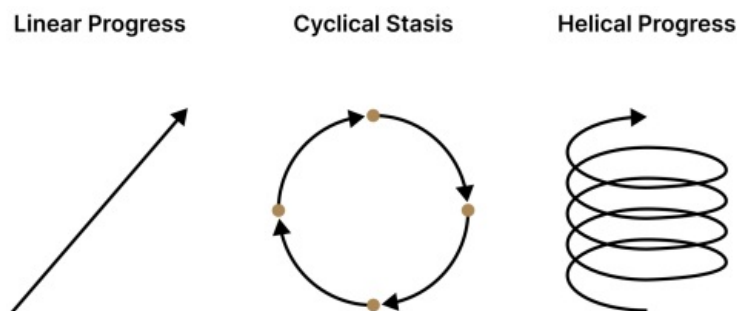
Balaji's helical theory of history: yes, there are cycles, but there is also ongoing progress. Right now, we're at the part of the cycle where we need to help the sclerotic old order die, but also seed a new and better one.

Do you have to agree with Balaji's megapolitics to like network states?

Many aspects of Balajian megapolitics will not be convincing to many readers. If you believe that "wokeness" is an important movement that protects the vulnerable, you may not appreciate the almost off-handed dismissal that it is basically just a mask for a professional elite's will-to-power. If you are worried about the plight of smaller countries such as Ukraine who are threatened by aggressive neighbors and desperately need outside support, you will not be convinced by Balaji's plea that "it may instead be best for countries to rearm, and take on their own defense".

I do think that you can support network states while disagreeing with some of Balaji's reasoning for them (and vice versa). But first, I should explain why I *think* Balaji feels that his view of the problem and his view of the solution are connected. Balaji has been passionate about roughly the same problem for a long time; you can see a similar narrative outline of defeating US institutional sclerosis through a technological and exit-driven approach in his speech on "the ultimate exit" from 2013. Network states are the latest iteration of his proposed solution.

There are a few reasons why talking about the problem is important:

- **To show that network states are the only way to protect freedom and capitalism, one must show why the US cannot.** If the US, or the "democratic liberal order", is just fine, then there is no need for alternatives; we should just double down on global coordination and rule of law. But if the US is in an irreversible decline, and its rivals are ascending, then things look quite different. Network states can "maintain liberal values in an illiberal world"; hegemony thinking that assumes "the good guys are in charge" cannot.
- **Many of Balaji's intended readers are not in the US, and a world of network states**

would inherently be globally distributed - and that includes *lots* of people who are suspicious of America. Balaji himself is Indian, and has a large Indian fan base. Many people in India, and elsewhere, view the US not as a "guardian of the liberal world order", but as something much more hypocritical at best and sinister at worst. Balaji wants to make it clear that you do not have to be pro-American to be a liberal (or at least a Balaji-liberal).

- **Many parts of US left-leaning media are increasingly hostile to both cryptocurrency and the tech sector**. Balaji expects that the "authoritarian left" parts of "team NYT" will be hostile to network states, and he explains this by pointing out that the media are not angels and their attacks are often self-interested.

But this is not the only way of looking at the broader picture. What if you *do* believe in the importance of role of social justice values, the New York Times, or America? What if you value governance innovation, but have more moderate views on politics? Then, there are two ways you could look at the issue:

- **Network states as a synergistic strategy, or at least as a backup**. Anything that happens in US politics in terms of improving *equality,* for example, only benefits the ~4% of the world's population that lives in the United States. The First Amendment does not apply outside US borders. The governance of many wealthy countries *is* sclerotic, and we do need *some* way to try more governance innovation. Network states could fill in the gaps. Countries like the United States could host network states that attract people from all over the world. Successful network states could even serve as a policy model for countries to adopt. Alternatively, *what if* the Republicans win and secure a decades-long majority in 2024, or the United States breaks down? You want there to be an alternative.
- **Exit to network states as a distraction, or even a threat**. If everyone's first instinct when faced with a large problem within their country is to exit to an enclave elsewhere, there will be no one left to protect and maintain the countries themselves. Global infrastructure that ultimately network states depend on will suffer.

Both perspectives are compatible with a lot of disagreement with Balajian megapolitics. Hence, to argue for or against Balajian network states, we will ultimately have to talk about network states. My own view is friendly to network states, though with a lot of caveats and different ideas about how network states could work.

What does cryptocurrency have to do with network states?

There are two kinds of alignment here: there is the *spiritual* alignment, the idea that ["Bitcoin becomes the flag of technology"](), and there is the *practical* alignment, the specific ways in which network states could use blockchains and cryptographic tokens. In general, I agree with both of these arguments - though I think Balaji's book could do much more to spell them out more explicitly.

The spiritual alignment

Cryptocurrency in 2022 is a key standard-bearer for internationalist liberal values that are difficult to find in any other social force that still stands strong today. Blockchains and cryptocurrencies are inherently global. Most Ethereum developers are outside the US, living in far-flung places like Europe, Taiwan and Australia. NFTs have given unique opportunities to [artists in Africa]() and [elsewhere in the Global South](). Argentinians punch above their weight in projects like Proof of Humanity, Kleros and Nomic Labs.

Blockchain communities continue to stand for openness, freedom, censorship resistance and [credible neutrality](), at a time where many geopolitical actors are increasingly only serving their own interests. This enhances their international appeal further: you don't have to love US hegemony to love blockchains and the values that they stand for. And this all makes blockchains an ideal spiritual companion for the network state vision that Balaji wants to see.

The practical alignment

But spiritual alignment means little without practical use value for blockchains to go along with it. Balaji gives plenty of blockchain use cases. One of Balaji's favorite concepts is the idea of the blockchain as a "ledger of record": people can timestamp events on-chain, creating a global provable log of humanity's "microhistory". He continues with other examples:

- [Zero-knowledge]() technology like [ZCash](), [Ironfish](), and [Tornado Cash]() allow on-chain attestation of exactly what people want to make public and nothing more.
- Naming systems like the [Ethereum Name Service]() (ENS) and [Solana Name Service]()

(SNS) attach identity to on-chain transactions.

- Incorporation systems allow the on-chain representation of corporate abstractions above the level of a mere transaction, like financial statements or even full programmable company-equivalents like DAOs.
- Cryptocredentials, Non-Fungible Tokens (NFTs), Non-Transferable Fungibles (NTFs), and Soulbounds allow the representation of non-financial data on chain, like diplomas or endorsements.

But how does this all relate to network states? I could go into specific examples in the vein of crypto cities: issuing tokens, issuing CityDAO-style citizen NFTs, combining blockchains with zero-knowledge cryptography to do secure privacy-preserving voting, and a lot more. Blockchains are the Lego of crypto-finance and crypto-governance: they are a very effective tool for implementing transparent in-protocol rules to govern common resources, assets and incentives.

But we also need to go a level deeper. **Blockchains and network states have the shared property that they are both trying to "create a new root"**. A corporation is not a root: if there is a dispute inside a corporation, it ultimately gets resolved by a national court system. Blockchains and network states, on the other hand, *are* trying to be new roots. This does not mean some absolute "na na no one can catch me" ideal of sovereignty that is perhaps only truly accessible to the ~5 countries that have highly self-sufficient national economies and/or nuclear weapons. Individual blockchain participants are of course vulnerable to national regulation, and enclaves of network states even more so. But blockchains are the only infrastructure system that at least *attempts* to do ultimate dispute resolution at the non-state level (either through on-chain smart contract logic or through the freedom to fork). This makes them an ideal base infrastructure for network states.

What aspects of Balaji's vision do I like?

Given that a purist "private property rights only" libertarianism inevitably runs into large problems like its inability to fund public goods, any successful pro-freedom program in the 21st century has to be a hybrid containing at least one Big Compromise Idea that solves at least 80% of the problems, so that independent individual initiative can take care of the rest. This could be some stringent measures against economic power and wealth concentration (maybe charge annual Harberger taxes on everything), it could be an 85% Georgist land tax, it could be a UBI, it could be mandating that sufficiently large companies become democratic internally, or one of any other proposals. Not all of these work, but you need *something* that drastic to have any shot at all.

Generally, I am used to the Big Compromise Idea being a leftist one: some form of equality and democracy. Balaji, on the other hand, has Big Compromise Ideas that feel more rightist: local communities with shared values, loyalty, religion, physical environments structured to encourage personal discipline ("keto kosher") and hard work. These values are implemented in a very libertarian and tech-forward way, organizing not around land, history, ethnicity and country, but around the cloud and personal choice, but they are rightist values nonetheless. This style of thinking is foreign to me, but I find it fascinating, and important. Stereotypical "wealthy white liberals" ignore this at their peril: these more "traditional" values are actually quite popular even among some ethnic minorities in the United States, and even more so in places like Africa and India, which is exactly where Balaji is trying to build up his base.

But what about this particular baizuo that's currently writing this review? Do network states actually interest me?

The "Keto Kosher" health-focused lifestyle immersion network state is certainly one that I would want to live in. Sure, I could just spend time in cities with lots of healthy stuff that I can seek out intentionally, but a concentrated physical environment makes it so much easier. Even the motivational aspect of being around other people who share a similar goal sounds very appealing.

But the truly interesting stuff is the governance innovation: using network states to organize in ways that would actually not be possible under existing regulations. There are three ways that you can interpret the underlying goal here:

1. Creating new **regulatory environments that let their residents have different priorities** from the priorities preferred by the mainstream: for example, the "anyone can walk around naked" zone, or a zone that implements different tradeoffs between safety and convenience, or a zone that legalizes more psychoactive substances.
2. Creating new **regulatory institutions that might be more efficient at serving the same priorities** as the status quo. For example, instead of improving environmental friendliness by regulating specific behaviors, you could just have a Pigovian tax. Instead of requiring licenses

and regulatory pre-approval for many actions, you could require [mandatory liability insurance](#). You could use [quadratic voting](#) for governance and quadratic funding to fund local public goods.
3. **Pushing against regulatory conservatism in general**, by increasing the chance that there's *some* jurisdiction that will let you do any particular thing. Institutionalized bioethics, for example, is a notoriously conservative enterprise, where 20 people dead in a medical experiment gone wrong is a tragedy, but 200000 people dead from [life-saving medicines and vaccines](#) not being approved quickly enough is a statistic. Allowing people to opt into network states that accept higher levels of risk could be a successful strategy for pushing against this.

In general, I see value in all three. A large-scale institutionalization of [1] could make the word simultaneously more free while making people comfortable with higher levels of restriction of certain things, because they know that if they want to do something disallowed there are other zones they could go to do it. More generally, I think there is an important idea hidden in [1]: **while the "social technology" community has come up with many good ideas around better *governance*, and many good ideas around better *[public discussion](#)*, there is a missing emphasis on better social technology for *sorting*.** We don't just want to take existing maps of social connections as given and find better ways to come to consensus within them. We also want to reform the webs of social connections themselves, and put people closer to other people that are more compatible with them to better allow different ways of life to maintain their own distinctiveness.

[2] is exciting because it fixes a major problem in politics: unlike startups, where the early stage of the process looks somewhat like a mini version of the later stage, in politics the early stage is a public discourse game that often selects for very different things than what actually work in practice. If governance ideas are regularly implemented in network states, then we would **move from an extrovert-privileging "talker liberalism" to a more balanced "doer liberalism" where ideas rise and fall based on how well they actually do on a small scale**. We could even combine [1] and [2]: have a zone for people who want to automatically participate in a new governance experiment every year as a lifestyle.

[3] is of course a more complicated moral question: whether you view paralysis and creep toward de-facto authoritarian global government as a bigger problem or someone inventing an evil technology that dooms us all as a bigger problem. I'm generally in the first camp; I am concerned about the prospect of [both the West and China settling into a kind of low-growth conservatism](#), I *love* how imperfect coordination between nation states limits the enforceability of things like global copyright law, and I'm concerned about the possibility that, with future surveillance technology, the world as a whole will enter a highly self-enforcing but terrible political equilibrium that it cannot get out of. But there are specific areas (cough cough, [unfriendly AI risk](#)) where I am in the risk-averse camp ... but here we're already getting into the second part of my reaction.

What aspects of Balaji's vision do I take issue with?

There are four aspects that I am worried about the most:

1. The "founder" thing - why do network states need a recognized founder to be so central?
2. What if network states end up only serving the wealthy?
3. "Exit" alone is not sufficient to stabilize global politics. So if exit is everyone's first choice, what happens?
4. What about global negative externalities more generally?

The "founder" thing

Throughout the book, Balaji is insistent on the importance of "founders" in a network state (or rather, a *startup society*: you found a startup society, and become a network state if you are successful enough to get diplomatic recognition). Balaji explicitly describes startup society founders as being "moral entrepreneurs":

> These presentations are similar to startup pitch decks. But as the founder of a startup society, you aren't a technology entrepreneur telling investors why this new innovation is better, faster, and cheaper. You are a moral entrepreneur telling potential future citizens about a better way of life, about a single thing that the broader world has gotten wrong that your community is setting right.

Founders crystallize moral intuitions and [learnings from history](#) into a concrete philosophy, and people whose moral intuitions are compatible with that philosophy coalesce around the project. This is all very reasonable at an early stage - though it is definitely not the *only* approach for how a startup society could emerge. But what happens at later stages? Mark Zuckerberg being the

centralized founder of facebook the startup was perhaps necessary. But Mark Zuckerberg being in charge of a multibillion-dollar (in fact, multibillion-*user*) company is something quite different. Or, for that matter, what about Balaji's nemesis: the fifth-generation hereditary white Ochs-Sulzberger dynasty running the New York Times?

Small things being centralized is great, extremely large things being centralized is terrifying. And given the reality of network effects, the freedom to exit again is not sufficient. In my view, the problem of how to settle into something other than founder control is important, and Balaji spends too little effort on it. "Recognized founder" is baked into the definition of what a Balajian network state is, but a roadmap toward wider participation in governance is not. It should be.

What about everyone who is not wealthy?

Over the last few years, we've seen many instances of governments around the world becoming explicitly more open to "tech talent". There are [42 countries offering digital nomad visas](#), there is a [French tech visa](#), a [similar program in Singapore](#), [golden visas for Taiwan](#), a program for [Dubai](#), and many others. This is all great for skilled professionals and rich people. Multimillionaires fleeing China's tech crackdowns and covid lockdowns (or, for that matter, moral disagreements with China's [other policies](#)) can often escape [the world's systemic discrimination against Chinese and other low-income-country citizens](#) by spending a few hundred thousand dollars on [buying another passport](#). But what about regular people? What about the [Rohingya minority facing extreme conditions in Myanmar](#), most of whom do not have a way to enter the US or Europe, much less buy another passport?

Here, we see a potential tragedy of the network state concept. On the one hand, I can really see how exit can be the most viable strategy for global human rights protection in the twenty first century. What do you do if another country is oppressing an ethnic minority? You could do nothing. You could sanction them (often [ineffective](#) and [ruinous](#) to the very people you're trying to help). You could try to invade (same criticism but even worse). Exit is a more humane option. People suffering human rights atrocities could just pack up and leave for friendlier pastures, and coordinating to do it in a group would mean that they could leave without sacrificing the communities they depend on for friendship and economic livelihood. And if you're wrong and the government you're criticizing is actually not that oppressive, then people won't leave and all is fine, no starvation or bombs required. This is all beautiful and good. Except... the whole thing breaks down because when the people try to exit, nobody is there to take them.

What is the answer? Honestly, I don't see one. One point in favor of network states is that they could be based *in* poor countries, and attract wealthy people from abroad who would then help the local economy. But this does nothing for people in poor countries who *want to get out*. Good old-fashioned political action within existing states to liberalize immigration laws seems like the only option.

Nowhere to run

In the wake of Russia's invasion of Ukraine on Feb 24, Noah Smith wrote an [important post](#) on the moral clarity that the invasion should bring to our thought. A particularly striking section is titled "nowhere to run". Quoting:

> But while exit works on a local level — if San Francisco is too dysfunctional, you can probably move to Austin or another tech town — it simply won't work at the level of nations. In fact, it never really did — rich crypto guys who moved to countries like Singapore or territories like Puerto Rico still depended crucially on the infrastructure and institutions of highly functional states. But Russia is making it even clearer that this strategy is doomed, because eventually there is nowhere to run. Unlike in previous eras, the arm of the great powers is long enough to reach anywhere in the world.

> If the U.S. collapses, you can't just move to Singapore, because in a few years you'll be bowing to your new Chinese masters. If the U.S. collapses, you can't just move to Estonia, because in a few years (months?) you'll be bowing to your new Russian masters. And those masters will have extremely little incentive to allow you to remain a free individual with your personal fortune intact ... Thus it is very very important to every libertarian that the U.S. not collapse.

One possible counter-argument is: sure, if Ukraine was full of people whose first instinct was exit, Ukraine would have collapsed. But if *Russia was also more exit-oriented*, everyone in Russia would have pulled out of the country within a week of the invasion. Putin would be left standing alone in the fields of the Luhansk oblast facing Zelensky a hundred meters away, and when Putin shouts his demand for surrender, Zelensky would reply: "you and what army"? (Zelensky would of course win a

fair one-on-one fight)

But things could go a different way. The risk is that exitocracy becomes recognized as *the primary way you do the "freedom" thing*, and societies that value freedom will become exitocratic, but centralized states will censor and suppress these impulses, adopt a militaristic attitude of national unconditional loyalty, and run roughshod over everyone else.

So what about those negative externalities?

If we have a hundred much-less-regulated innovation labs everywhere around the world, this could lead to a world where harmful things are more difficult to prevent. This raises a question: **does believing in Balajism *require* believing in a world where negative externalities are *not too big* a deal?** Such a viewpoint would be *the opposite* of the [Vulnerable World Hypothesis (VWH)](#), which suggests that are technology progresses, it gets easier and easier for one or a few crazy people to kill millions, and global authoritarian surveillance might be *required* to prevent extreme suffering or even extinction.

One way out might be to focus on self-defense technology. Sure, in a network state world, we could not feasibly ban gain-of-function research, but we could use network states to help the world along a path to adopting really good [HEPA air filtering](#), [far-UVC light](#), early detection [infrastructure](#) and a very rapid vaccine development and deployment pipeline that could defeat not only covid, but far worse viruses too. [This 80,000 hours episode](#) outlines the bull case for bioweapons being a solvable problem. But this is not a universal solution for all technological risks: at the very least, there is no self-defense against a super-intelligent unfriendly AI that kills us all.

Self-defense technology is good, and is probably an undervalued funding focus area. But it's not realistic to rely on that alone. Transnational cooperation to, for example, [ban slaughterbots](#), would be required. And so we do want a world where, even if network states have more sovereignty than intentional communities today, their sovereignty is not *absolute*.

Non-Balajian network states

Reading *The Network State* reminded me of a different book that I read ten years ago: David de Ugarte's [*Phyles: Economic Democracy in the Twenty First Century*](#). *Phyles* talks about similar ideas of transnational communities organized around values, but it has a much more left-leaning emphasis: it assumes that these communities will be democratic, inspired by a combination of 2000s-era online communities and nineteenth and twentieth-century ideas of cooperatives and workplace democracy.

We can see the differences most clearly by looking at de Ugarte's theory of formation. Since I've already spent a lot of time quoting Balaji, I'll give de Ugarte a fair hearing with a longer quote:

> The very blogosphere is an ocean of identities and conversation in perpetual cross-breeding and change from among which the great social digestion periodically distils stable groups with their own contexts and specific knowledge.
>
> These conversational communities which crystallise, after a certain point in their development, play the main roles in what we call digital Zionism: they start to precipitate into reality, to generate mutual knowledge among their members, which makes them more identitarially important to them than the traditional imaginaries of the imagined communities to which they are supposed to belong (nation, class, congregation, etc.) as if it were a real community (group of friends, family, guild, etc.)
>
> Some of these conversational networks, identitarian and dense, start to generate their own economic metabolism, and with it a distinct demos – maybe several demoi – which takes the nurturing of the autonomy of the community itself as its own goal. These are what we call Neo-Venetianist networks. Born in the blogosphere, they are heirs to the hacker work ethic, and move in the conceptual world, which tends to the economic democracy which we spoke about in the first part of this book.
>
> Unlike traditional cooperativism, as they do not spring from real proximity-based communities, their local ties do not generate identity. In the Indianos' foundation, for instance, there are residents in two countries and three autonomous regions, who started out with two companies founded hundreds of kilometres away from each other.

We see some very Balajian ideas: shared collective identities, but formed around values rather than geography, that start off as discussion communities in the cloud but then materialize into taking over large portions of economic life. De Ugarte even uses the exact same metaphor ("digital Zionism") that

Balaji does!

But we also see a key difference: there is no single founder. Rather than a startup society being formed by an act of a single individual combining together intuitions and strands of thought into a coherent formally documented philosophy, a phyle starts off as a conversational network in the blogosphere, and then directly turns into a group that does more and more over time - all while keeping its democratic and horizontal nature. The whole process is much more organic, and not at all guided by a single person's intention.

Of course, the immediate challenge that I can see is the incentive issues inherent to such structures. One way to perhaps unfairly summarize both *Phyles* and *The Network State* is that *The Network State* seeks to use 2010s-era blockchains as a model for how to reorganize human society, and *Phyles* seeks to use 2000s-era open source software communities and blogs as a model for how to reorganize human society. Open source has the failure mode of *not enough incentives*, cryptocurrency has the failure mode of *excessive and overly concentrated incentives*. But what this does suggest is that some kind of middle way should be possible.

Is there a middle way?

My judgement so far is that network states are great, but they are far from being a viable Big Compromise Idea that can actually plug all the holes needed to build the kind of world I and most of my readers would want to see in the 21st century. Ultimately, I do think that we need to bring in more democracy and large-scale-coordination oriented Big Compromise Ideas of some kind to make network states truly successful.

Here are some significant adjustments to Balajism that I would endorse:

Founder to start is okay (though not the only way), but we really need a baked-in roadmap to exit-to-community

Many founders *want* to eventually retire or start something new (see: basically half of every crypto project), and we need to prevent network states from collapsing or sliding into mediocrity when that happens. Part of this process is some kind of constitutional **exit-to-community** guarantee: as the network state enters higher tiers of maturity and scale, more input from community members is taken into account automatically.

Prospera attempted something like this. As Scott Alexander summarizes:

> Once Próspera has 100,000 residents (so realistically a long time from now, if the experiment is very successful), they can hold a referendum where 51% majority can change anything about the charter, including kicking HPI out entirely and becoming a direct democracy, or rejoining the rest of Honduras, or anything

But I would favor something even more participatory than the residents having an all-or-nothing nuclear option to kick the government out.

Another part of this process, and one that I've recognized in the process of Ethereum's growth, is explicitly encouraging broader participation in the moral and philosophical development of the community. Ethereum has its Vitalik, but it also has its Polynya: an internet anon who has recently entered the scene unsolicited and started providing high-quality thinking on rollups and scaling technology. How will your startup society recruit its first ten Polynyas?

Network states should be run by something that's not coin-driven governance

Coin-driven governance is plutocratic and vulnerable to attacks; I have written about this many times, but it's worth repeating. Ideas like Optimism's soulbound and one-per-person citizen NFTs are key here. Balaji already acknowledges the need for non-fungibility (he supports coin lockups), but we should go further and more explicit in supporting governance that's not just shareholder-driven. This will also have the beneficial side effect that more democratic governance is more likely to be aligned with the outside world.

Network states commit to making themselves friendly through outside representation in governance

One of the fascinating and under-discussed ideas from the rationalist and friendly-AI community is

functional decision theory. This is a complicated concept, but the powerful core idea is that AIs could coordinate better than humans, solving prisoner's dilemmas where humans often fail, by making verifiable public commitments about their source code. An AI could rewrite itself to have a module that prevents it from cheating other AIs that have a similar module. Such AIs would all cooperate with each other [in prisoner's dilemmas](#).

As I [pointed out years ago](#), DAOs could potentially do the same thing. They could have governance mechanisms that are explicitly more charitable toward other DAOs that have a similar mechanism. Network states would be run by DAOs, and this would apply to network states too. They could even commit to governance mechanisms that promise to take wider public interests into account (eg. 20% of the votes could go to a randomly selected set of residents of the host city or country), without the burden of having to follow specific complicated regulations of *how* they should take those interests into account. A world where network states do such a thing, and where countries adopt policies that are explicitly more friendly to network states that do it, could be a better one.

Conclusion

I want to see startup societies along these kinds of visions exist. I want to see immersive lifestyle experiments around healthy living. I want to see crazy governance experiments where public goods are funded by quadratic funding, and all zoning laws are replaced by a system where every building's property tax floats between zero and five percent per year based on what percentage of nearby residents express approval or disapproval in a real-time [blockchain and ZKP-based voting system](#). And I want to see more technological experiments that accept higher levels of risk, if the people taking those risks consent to it. And I think blockchain-based tokens, identity and reputation systems and DAOs could be a great fit.

At the same time, I worry that the network state vision in its current form risks only satisfying these needs for those wealthy enough to move and desirable enough to attract, and many people lower down the socioeconomic ladder will be left in the dust. What can be said in network states' favor is their internationalism: we even have the Africa-focused [Afropolitan](#). Inequalities *between countries* are [responsible for two thirds of global inequality](#) and inequalities *within* countries are only one third. But that still leaves a lot of people in all countries that this vision doesn't do much for. So we need something else too - for the global poor, for Ukrainians that want to keep their country and not just squeeze into Poland for a decade until Poland gets invaded too, and everyone else that's not in a position to move to a network state tomorrow or get accepted by one.

Network states, with some modifications that push for more democratic governance and positive relationships with the communities that surround them, *plus* some other way to help everyone else? That is a vision that I can get behind.

My 40-liter backpack travel guide

2022 Jun 20

Special thanks to Liam Horne for feedback and review. I received no money from and have never even met any of the companies making the stuff I'm shilling here (with the sole exception of Unisocks); this is all just an honest listing of what works for me today.

I have lived as a nomad for the last nine years, taking 360 flights travelling over 1.5 million kilometers (assuming flight paths are straight, ignoring layovers) during that time. During this time, I've considerably optimized the luggage I carry along with me: from a 60-liter shoulder bag with a separate laptop bag, to a 60-liter shoulder bag that can contain the laptop bag, and now to a 40-liter packpage that can contain the laptop bag along with all the supplies I need to live my life.

The purpose of this post will be to go through the contents, as well as some of the tips that I've learned for how you too can optimize your travel life and never have to wait at a luggage counter again. There is no obligation to follow this guide in its entirety; if you have important needs that differ from mine, you can still get a lot of the benefits by going a hybrid route, and I will talk about these options too.

This guide is focused on my own experiences; plenty of other people have made their own guides and you should look at them too. /r/onebag is an excellent subreddit for this.

The backpack, with the various sub-bags laid out separately. Yes, this all fits in the backpack, and without that much effort to pack and unpack.

As a point of high-level organization, notice the bag-inside-a-bag structure. I have a T-shirt bag, an underwear bag, a sock bag, a toiletries bag, a dirty-laundry bag, a medicine bag, a laptop bag, and various small bags inside the inner compartment of my backpack, which all fit into a [40-liter Hynes Eagle backpack](#). This structure makes it easy to keep things organized.

It's like frugality, but for cm^3 instead of dollars

The general principle that you are trying to follow is that you're trying to stay within a "budget" while still making sure you have everything that you need - much like normal financial planning of the type that almost everyone, with the important exception of crypto participants during bull runs, is used to dealing with. A key difference here is that instead of optimizing for dollars, you're optimizing for *cubic centimeters*. Of course, none of the things that I recommend here are going to be particularly hard on your dollars either, but minimizing cm^3 is the primary objective.

What do I mean by this? Well, I mean getting items like this:

Electric shaver. About 5cm long and 2.5cm wide at the top. No charger or handle is required: it's USBC pluggable, your phone is the charger and handle. Buy [on Amazon here](#) (told you it's not hard on your dollars!)

And this:

Charger for mobile phone and laptop (can charge both at the same time)! About 5x5x2.5 cm. Buy [here](#).

And there's more. Electric toothbrushes are normally known for being wide and bulky. But they don't have to be! [Here](#) is an electric toothbrush that is rechargeable, USBC-friendly (so no extra charging equipment required), only slightly wider than a regular toothbrush, and costs about $30, plus a couple dollars every few months for replacement brush heads. For connecting to various different continents' plugs, you can either use [any regular reasonably small universal adapter](#), or get the [Zendure Passport III](#) which combines a universal adapter with a charger, so you can plug in USBC cables to charge your laptop and multiple other devices directly (!!).

As you might have noticed, **a key ingredient in making this work is to be a USBC maximalist. You should strive to ensure that every single thing you buy is USBC-friendly.** Your laptop, your phone, your toothbrush, everything. This ensures that you don't need to carry any extra equipment beyond one charger and 1-2 charging cables. In the last ~3 years, it has become much easier to live the USBC maximalist life; enjoy it!

Be a Uniqlo maximalist

For clothing, you have to navigate a tough tradeoff between price, cm^3 and the clothing looking reasonably good. Fortunately, many of the more modern brands do a great job of fulfilling all three at the same time! My current strategy is to be a Uniqlo maximalist: altogether, about 70% of the clothing items in my bag are from Uniqlo.

This includes:

- 8 T-shirts, of which 6 are [this type from Uniqlo](#)
- 8 pairs of underwear, mostly various Uniqlo products
- 8 socks, of which none are Uniqlo (I'm less confident about what to do with socks than with other clothing items, more on this later)
- Heat-tech [tights](#), from Uniqlo
- Heat-tech sweater, from Uniqlo
- Packable jacket, from Uniqlo
- Shorts that also double as a swimsuit, from.... ok fine, it's also Uniqlo.

There are other stores that can give you often equally good products, but Uniqlo is easily accessible in many (though not all) of the regions I visit and does a good job, so I usually just start and stop there.

Socks

Socks are a complicated balancing act between multiple desired traits:

- Low cm^3
- Easy to put on
- Warm (when needed)
- Comfortable

The ideal scenario is if you find [low-cut or ankle socks](#) comfortable to wear, and you never go to cold climates. These are very low on cm^3, so you can just buy those and be happy. But this doesn't work for me: I sometimes visit cold areas, I don't find ankle socks comfortable and prefer something a bit longer, and I need to be comfortable for my long runs. Furthermore, my large foot size means that Uniqlo's one-size-fits-all approach does not work well for me: though I can put the socks on, it often takes a long time to do so (especially after a shower), and the socks rip often.

So I've been exploring various brands to try to find a solution (recently trying [CEP](#) and [DarnTough](#)). I generally try to find socks that cover the ankle but don't go much higher than that, and I have one pair of long ones for when I go to the snowier places. My sock bag is currently larger than my underwear bag, and only a bit smaller than my T-shirt bag: both a sign of the challenge of finding good socks, and a testament to Uniqlo's amazing Airism T-shirts. Once you do find a pair of socks that you like, ideally you should just buy many copies of the same type. This removes the effort of searching for a matching pair in your bag, and it ensures that if one of your socks rips you don't have to choose between losing the whole pair and wearing mismatched socks.

For shoes, you probably want to limit yourself to at most two: some heavier shoes that you can just wear, and some very cm^3-light alternative, such as flip-flops.

Layers

There is a key mathematical reason why dressing in layers is a good idea: it lets you cover many possible temperature ranges with fewer clothing items.

Temperature (°C)	Clothing

20°

13°   +
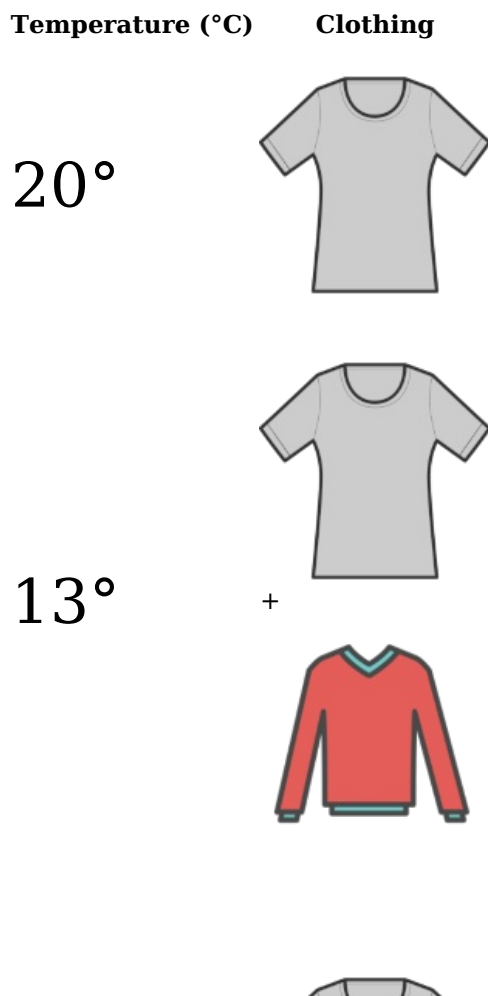
7°

+

0°

+

+

You want to keep the T-shirt on in all cases, to protect the other layers from getting dirty. But aside from that, the general rule is: if you choose N clothing items, with levels of warmness spread out across powers of two, then you can be comfortable in 2^N different temperature ranges by binary-encoding the expected temperature in the clothing you wear. For not-so-cold climates, two layers (sweater and jacket) are fine. For a more universal range of climates you'll want three layers: light sweater, heavy sweater and heavy jacket, which can cover $2^3 = 8$ different temperature ranges all the way from summer to Siberian winter (of course, heavy winter jackets are not easily packable, so you may have to just wear it when you get on the plane).

This layering principle applies not just to upper-wear, but also pants. I have a pair of thin pants plus Uniqlo tights, and I can wear the thin pants alone in warmer climates and put the Uniqlo tights under them in colder climates. The tights also double as pyjamas.

My miscellaneous stuff

The internet constantly yells at me for not having a good microphone. I solved this problem by getting a portable microphone!

My workstation, using the [Apogee HypeMIC](#) travel microphone (unfortunately micro-USB, not USBC). A toilet paper roll works great as a stand, but I've also found that having a stand is not really necessary and you can just let the microphone lie down beside your laptop.

Next, my **laptop stand**. Laptop stands are great for improving your posture. I have two recommendations for laptop stands, one medium-effective but very light on cm^3, and one very effective but heavier on cm^3.

- The lighter one: **Majextand**
- The more powerful one: **Nexstand**

Nexstand is the one in the picture above. Majextand is the one glued to the bottom of my laptop now:

I have used both, and recommend both. In addition to this I also have another piece of laptop gear: a [20000 mAh laptop-friendly power bank](). This adds even more to my laptop's already decent battery life, and makes it generally easy to live on the road.

Now, my **medicine bag**:

This contains a combination of various life-extension medicines (metformin, ashwagandha, and some vitamins), and covid defense gear: a CO2 meter (CO2 concentration minus 420 roughly gives you how much human-breathed-out air you're breathing in, so it's a good proxy for virus risk), masks, antigen tests and fluvoxamine. The tests were a free care package from the Singapore government, and they happened to be excellent on cm^3 so I carry them around. Covid defense and life extension are both fields where the science is rapidly evolving, so don't blindly follow this static list; follow the science yourself or listen to the latest advice of an expert that you do trust. Air filters and far-UVC (especially 222 nm) lamps are also promising covid defense options, and portable versions exist for both.

At this particular time I don't happen to have a first aid kit with me, but in general it's also recommended; plenty of good travel options exist, eg. [this]().

Finally, **mobile data**. Generally, you want to make sure you have a phone that supports eSIM. These days, more and more phones do. Wherever you go, you can buy an eSIM for that place online. I personally use [Airalo](), but there are many options. If you are lazy, you can also just use [Google Fi](), though in my experience Google Fi's quality and reliability of service tends to be fairly mediocre.

Have some fun!

Not everything that you have needs to be designed around cm^3 minimization. For me personally, I have four items that are not particularly cm^3 optimized but that I still really enjoy having around.

My laptop bag, bought in an outdoor market in Zambia.

Unisocks.

Sweatpants for indoor use, that are either fox-themed or Shiba Inu-themed depending on whom you ask.

Gloves (phone-friendly): I bought the left one for $4 in Mong Kok and the right one for $5 in Chinatown, Toronto back in 2016. By coincidence, I lost different ones from each pair, so the remaining two match. I keep them around as a reminder of the time when money was much more scarce for me.

The more you save space on the boring stuff, the more you can leave some space for a few special items that can bring the most joy to your life.

How to stay sane as a nomad

Many people find the nomad lifestyle to be disorienting, and report feeling comfort from having a "permanent base". I find myself not really having these feelings: I do feel disorientation when I change locations more than once every ~7 days, but as long as I'm in the same place for longer than that, I acclimate and it "feels like home". I can't tell how much of this is my unique difficult-to-replicate personality traits, and how much can be done by anyone. In general, some tips that I recommend are:

- **Plan ahead**: make sure you know where you'll be at least a few days in advance, and know where you're going to go when you land. This reduces feelings of uncertainty.
- **Have some other regular routine**: for me, it's as simple as having a piece of dark chocolate and a cup of tea every morning (I prefer [Bigelow green tea decaf](), specifically the 40-packs, both because it's the most delicious decaf green tea I've tried and because it's packaged in a four-teabag-per-bigger-bag format that makes it very convenient and at the same time cm^3-friendly). Having *some* part of your lifestyle the same every day helps me feel grounded. The more digital your life is, the more you get this "for free" because you're staring into the same computer no matter what physical location you're in, though this does come at the cost of nomadding potentially providing fewer *benefits*.
- **Your nomadding should be embedded in some community**: if you're just being a lowest-common-denominator tourist, you're doing it wrong. Find people in the places you visit who have some key common interest (for me, of course, it's blockchains). Make friends in different cities. This helps you learn about the places you visit and gives you an understanding of the local culture in a way that "ooh look at the 800 year old statue of the emperor" never will. Finally, find other nomad friends, and make sure to intersect with them regularly. If home can't be a single place, home can be the people you jump places with.
- **Have some semi-regular bases**: you don't have to keep visiting a completely new location every time. Visiting a place that you have seen before reduces mental effort and adds to the feeling of regularity, and having places that you visit frequently gives you opportunities to put stuff down, and is important if you want your friendships and local cultural connections to actually develop.

How to compromise

Not everyone can survive with just the items I have. You might have some need for heavier clothing that cannot fit inside one backpack. You might be a big nerd in some physical-stuff-dependent field: I know life extension nerds, covid defense nerds, and many more. You might really love your three monitors and keyboard. You might have children.

The 40-liter backpack is in my opinion a truly ideal size if you *can* manage it: 40 liters lets you carry a week's worth of stuff, and generally all of life's basic necessities, and it's at the same time very carry-friendly: I have never had it rejected from carry-on in all the flights on many kinds of airplane that I have taken it, and when needed I can just barely stuff it under the seat in front of me in a way that looks legit to staff. Once you start going lower than 40 liters, the disadvantages start stacking up and exceeding the marginal upsides. But if 40 liters is not enough for you, there are two natural fallback options:

- **A larger-than-40 liter backpack**. You can find [50 liter backpacks](), [60 liter backpacks]() or [even larger]() (I highly recommend backpacks over shoulder bags for carrying friendliness). But the higher you go, the more tiring it is to carry, the more risk there is on your spine, and the more you incur the risk that you'll have a difficult situation bringing it as a carry-on on the plane and might even have to check it.
- **Backpack plus mini-suitcase.** There are plenty of [carry-on suitcases]() that you can buy. You can often make it onto a plane with a backpack *and* a mini-suitcase. This depends on you: you may find this to be an easier-to-carry option than a really big backpack. That said, there is sometimes a risk that you'll have a hard time carrying it on (eg. if the plane is very full) and occasionally you'll have to check something.

Either option can get you up to a respectable 80 liters, and still preserve *a lot* of the benefits of the 40-liter backpack lifestyle. Backpack plus mini-suitcase generally seems to be more popular than the big backpack route. It's up to you to decide which tradeoffs to take, and where your personal values lie!

Some ways to use ZK-SNARKs for privacy

2022 Jun 15

*Special thanks to Barry Whitehat and Gubsheep for feedback and review.*

ZK-SNARKs are a powerful cryptographic tool, and an increasingly important part of the applications that people are building both in the blockchain space and beyond. But they are complicated, both in terms of *how they work*, and in terms of *how you can use them*.

My [previous post explaining ZK-SNARKs](#) focused on the first question, attempting to explain the math behind ZK-SNARKs in a way that's reasonably understandable but still theoretically complete. This post will focus on the second question: how do ZK-SNARKs fit into existing applications, what are some examples of what they can do, what can't they do, and what are some general guidelines for figuring out whether or not ZK-SNARKing some particular application is possible?

In particular, this post focuses on applications of ZK-SNARKs for preserving privacy.

What does a ZK-SNARK do?

Suppose that you have a public input x, a private input w, and a (public) function $f(x, w) \rightarrow \{True, False\}$ that performs some kind of verification on the inputs. With a ZK-SNARK, you can prove that you know an w such that $f(x, w) = True$ for some given f and x, without revealing what w is. Additionally, the verifier can verify the proof much faster it would take for them to compute $f(x, w)$ themselves, even if they know w.
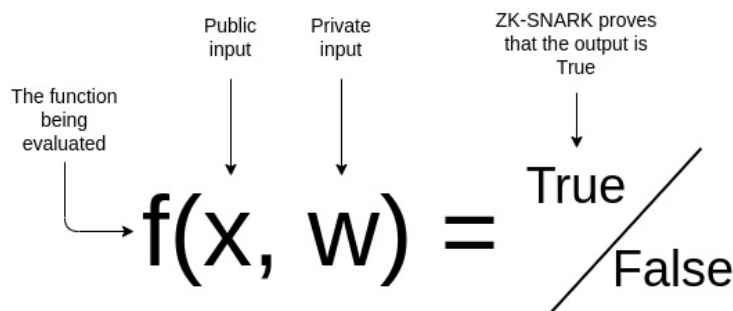
This gives the ZK-SNARK its two properties: **privacy** and **scalability**. As mentioned above, in this post our examples will focus on privacy.

Proof of membership

Suppose that you have an Ethereum wallet, and you want to prove that this wallet has a proof-of-humanity registration, without revealing *which* registered human you are. We can mathematically describe the function as follows:

- The **private input** (w): your address A, and the private key k to your address
- The **public input** (x): the set of all addresses with verified proof-of-humanity profiles $\{H_1 ... H_n\}$
- The **verification function** $f(x, w)$:
    - Interpret w as the pair (A, k), and x as the list of valid profiles $\{H_1 ... H_n\}$
    - Verify that A is one of the addresses in $\{H_1 ... H_n\}$
    - Verify that $privtoaddr(k) = A$
    - Return $True$ if both verifications pass, $False$ if either verification fails

The prover generates their address A and the associated key k, and provides $w = (A, k)$ as the private input to f. They take the public input, the current set of verified proof-of-humanity profiles $\{H_1 ... H_n\}$, from the chain. They run the ZK-SNARK proving algorithm, which (assuming the inputs are correct) generates the proof. The prover sends the proof to the verifier and they provide the block height at which they obtained the list of verified profiles.

The verifier also reads the chain, gets the list $\{H_1 ... H_n\}$ at the height that the prover specified, and checks the proof. If the check passes, the verifier is convinced that the prover has *some* verified proof-of-

humanity profile.

Before we move on to more complicated examples, I highly recommend you go over the above example until you understand every bit of what is going on.

Making the proof-of-membership more efficient

One weakness in the above proof system is that the verifier needs to know the whole set of profiles $\{H_1 ... H_n\}$, and they need to spend $O(n)$ time "inputting" this set into the ZK-SNARK mechanism.

We can solve this by instead passing in as a public input an on-chain Merkle root containing *all* profiles (this could just be the state root). We add another private input, a Merkle proof M proving that the prover's account A is in the relevant part of the tree.
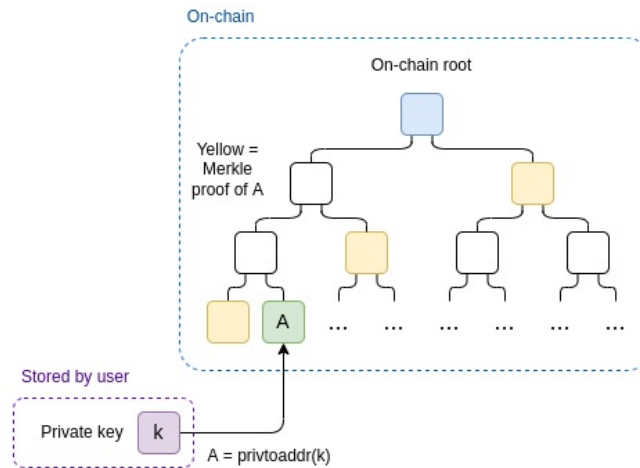
Advanced readers: A very new and more efficient alternative to Merkle proofs for ZK-proving membership is [Caulk](#). In the future, some of these use cases may migrate to Caulk-like schemes.

ZK-SNARKs for coins

Projects like [Zcash](#) and [Tornado.cash](#) allow you to have privacy-preserving *currency*. Now, you might think that you can take the "ZK proof-of-humanity" above, but instead of proving access of a proof-of-humanity profile, use it to prove access to a *coin*. But we have a problem: we have to *simultaneously solve privacy and the double spending problem*. That is, it should not be possible to spend the coin twice.

Here's how we solve this. Anyone who has a coin has a private secret s. They locally compute the "leaf" $L = hash(s, 1)$, which gets published on-chain and becomes part of the state, and $N = hash(s, 2)$, which we call the *nullifier*. The state gets stored in a Merkle tree.
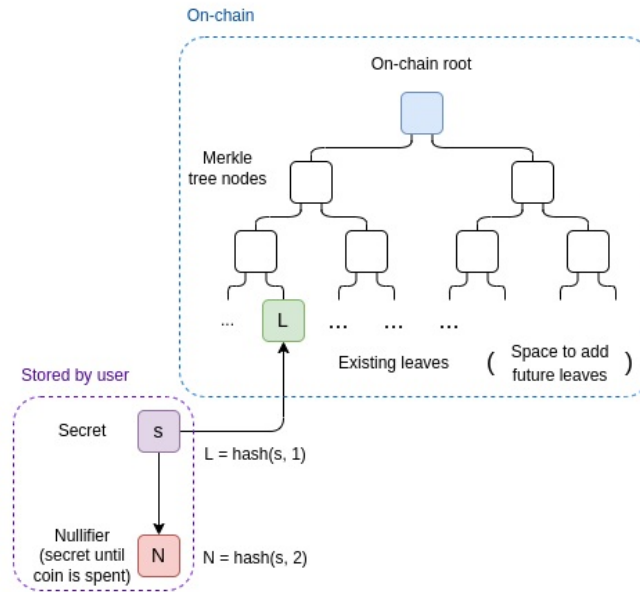
To spend a coin, the sender must make a ZK-SNARK where:

- The **public input** contains a nullifier N, the current or recent Merkle root R, and a new leaf L' (the intent is that recipient has a secret s', and passes to the sender $L' = hash(s', 1)$)
- The **private input** contains a secret s, a leaf L and a Merkle branch M
- The **verification function** checks that:
    - M is a valid Merkle branch proving that L is a leaf in a tree with root R, where R is the current Merkle root of the state
    - $hash(s, 1) = L$
    - $hash(s, 2) = N$

The transaction contains the nullifier N and the new leaf L'. We don't actually prove anything about L', but we "mix it in" to the proof to prevent L' from being modified by third parties when the transaction is in-flight.

To verify the transaction, the chain checks the ZK-SNARK, and additionally checks that N has not been used in a previous spending transaction. If the transaction succeeds, N is added to the spent nullifier set, so that it cannot be spent again. L' is added to the Merkle tree.

What is going on here? We are using a zk-SNARK to relate two values, L (which goes on-chain when a coin is created) and N (which goes on-chain when a coin is spent), without revealing *which* L is connected to *which* N. The connection between L and N can only be discovered if you know the secret s that generates both. Each coin that gets created can only be spent once (because for each L there is only one valid corresponding N), but *which* coin is being spent at a particular time is kept hidden.

This is also an important primitive to understand. Many of the mechanisms we describe below will be based on a very similar "privately spend only once" gadget, though for different purposes.

Coins with arbitrary balances

The above can easily be extended to coins of arbitrary balances. We keep the concept of "coins", except each coin has a (private) balance attached. One simple way to do this is have the chain store for each coin not just the leaf L but also an encrypted balance.

Each transaction would consume *two* coins and create two new coins, and it would add two (leaf, encrypted balance) pairs to the state. The ZK-SNARK would also check that the sum of the balances coming in equals the sum of the balances going out, and that the two output balances are both non-negative.

ZK anti-denial-of-service

An interesting anti-[denial-of-service](#) gadget. Suppose that you have some on-chain identity that is non-trivial to create: it could be a proof-of-humanity profile, it could be a validator with 32 ETH, or it could just be an account that has a nonzero ETH balance. We could create a more DoS resistant peer-to-peer network by only accepting a message if it comes with a proof that the message's sender has such a profile. Every profile would be allowed to send up to 1000 messages per hour, and a sender's profile would be removed from the list if the sender cheats. But how do we make this privacy-preserving?

First, the setup. Let k be the private key of a user; $A = \text{privtoaddr}(k)$ is the corresponding address. The list of valid addresses is public (eg. it's a registry on-chain). So far this is similar to the proof-of-humanity example: you have to prove that you have the private key to one address without revealing which one. But here, we don't just want a proof that you're in the list. We want a protocol that lets you prove you're in the list *but prevents you from making too many proofs*. And so we need to do some more work.

We'll divide up time into epochs; each epoch lasts 3.6 seconds (so, 1000 epochs per hour). Our goal will be to allow each user to send only one message per epoch; if the user sends *two* messages in the same epoch, they will get caught. To allow users to send occasional bursts of messages, they are allowed to use epochs in the recent past, so if some user has 500 unused epochs they can use those epochs to send 500 messages all at once.

The protocol

We'll start with a simple version: we use nullifiers. A user generates a nullifier with $N = \text{hash}(k, e)$, where k is their key and e is the epoch number, and publishes it along with the message m. The ZK-SNARK once again mixes in $\text{hash}(m)$ without verifying anything about m, so that the proof is bound to a single message. If a user makes two proofs bound to two different messages with the same nullifier, they can get caught.

Now, we'll move on to the more complex version. Instead of just making it easy to prove if someone used the same epoch twice, this next protocol will actually *reveal their private key* in that case. Our core technique will rely on the "two points make a line" trick: if you reveal one point on a line, you've revealed little, but if you reveal two points on a line, you've revealed the whole line.

For each epoch e, we take the line $L_e(x) = \text{hash}(k, e) * x + k$. The slope of the line is $\text{hash}(k, e)$, and the y-intercept is k; neither is known to the public. To make a *certificate* for a message m, the sender provides $y = L_e(\text{hash}(m)) =$ $\text{hash}(k, e) * \text{hash}(m) + k$, along with a ZK-SNARK proving that y was computed correctly.
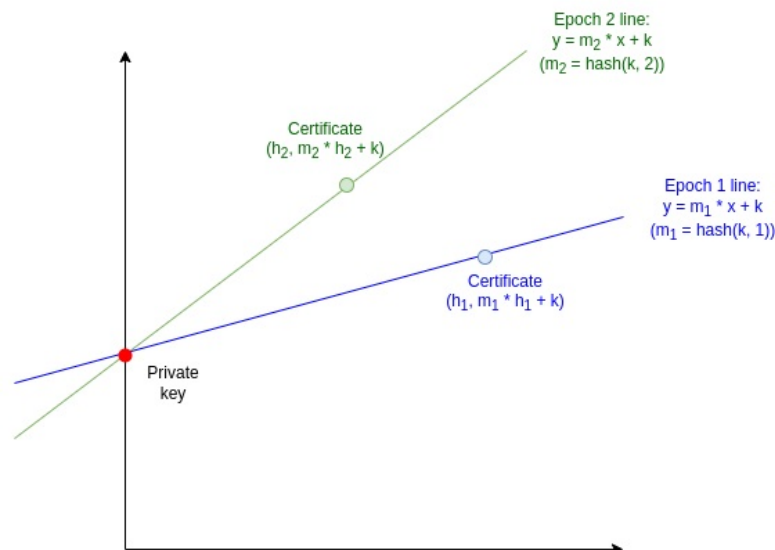
To recap, the ZK-SNARK here is as follows:

- **Public input**:
    - $\{A\_1 ... A\_n\}$, the list of valid accounts
    - m, the message that the certificate is verifying
    - e, the epoch number used for the certificate
    - y, the line function evaluation
- **Private input**:
    - k, your private key
- **Verification function**:
    - Check that $\text{privtoaddr}(k)$ is in $\{A\_1 ... A\_n\}$
    - Check that $y = \text{hash}(k, e) * \text{hash}(m) + k$

But what if someone uses a single epoch twice? That means they published two values m_1 and m_2 and the corresponding certificate values $y\_1 = \text{hash}(k, e) * \text{hash}(m\_1) + k$ and $y\_2 = \text{hash}(k, e) * \text{hash}(m\_2) + k$. We can use the two points to recover the line, and hence the y-intercept (which is the private key):

$$k = y_1 - hash(m_1) * \frac{y_2 - y_1}{hash(m_2) - hash(m_1)}$$

So if someone reuses an epoch, they leak out their private key for everyone to see. Depending on the circumstance, this could imply stolen funds, a slashed validator, or simply the private key getting broadcasted and included into a smart contract, at which point the corresponding address would get removed from the set.

What have we accomplished here? A viable off-chain, anonymous anti-denial-of-service system useful for systems like blockchain peer-to-peer networks, chat applications, etc, without requiring any proof of work. The [RLN (rate limiting nullifier)](#) project is currently building essentially this idea, though with minor modifications (namely, they do *both* the nullifier and the two-points-on-a-line technique, using the nullifier to make it easier to catch double-use of an epoch).

ZK negative reputation

Suppose that we want to build **0chan**, an internet forum which provides full anonymity like 4chan (so you don't even have persistent names), but has a reputation system to encourage more quality content. This could be a system where some moderation DAO can flag posts as violating the rules of the system and institutes a three-strikes-and-you're-out mechanism, it could be users being able to upvote and downvote posts; there are lots of configurations.

The reputation system could support positive or negative reputation; however, supporting negative reputation requires extra infrastructure to *require* the user to take into account all reputation messages in their proof, even the negative ones. It's this harder use case, which is similar to what is being implemented with [Unirep Social](#), that we'll focus on.

Chaining posts: the basics

Anyone can make a post by publishing a message on-chain that contains the post, and a ZK-SNARK proving that either (i) you own some scarce external identity, eg. proof-of-humanity, that entitles you to create an account, or (ii) that you made some specific previous post. Specifically, the ZK-SNARK is as follows:

- **Public inputs**:
  - The nullifier N
  - A recent blockchain state root R
  - The post contents ("mixed in" to the proof to bind it to the post, but we don't do any computation on it)
- **Private inputs**:
  - Your private key k
  - Either an external identity (with address A), or the nullifier N_{prev} used by the previous post
  - A Merkle proof M proving inclusion of A or N_{prev} on-chain
  - The number i of posts that you have previously made with this account
- **Verification function**:
  - Check that M is a valid Merkle branch proving that (either A or N_{prev}, whichever is provided) is a leaf in a tree with root R
  - Check that $N = enc(i, k)$, where enc is an encryption function (eg. AES)
  - If $i = 0$, check that $A = privtoaddr(k)$, otherwise check that $N_{prev} = enc(i-1, k)$

In addition to verifying the proof, the chain also checks that (i) R actually is a recent state root, and (ii) the nullifier N has not yet been used. So far, this is like the privacy-preserving coin introduced earlier, but we add a procedure for "minting" a new account, and we remove the ability to "send" your account to a different key - instead, all nullifiers are generated using your original key.

We use enc instead of $hash$ here to make the nullifiers reversible: if you have k, you can decrypt any specific nullifier you see on-chain and if the result is a valid index and not random junk (eg. we could just check $dec(N) < 2^{64}$), then you know that nullifier was generated using k.

Adding reputation

Reputation in this scheme is on-chain and in the clear: some smart contract has a method `addReputation`, which takes as input (i) the nullifier published along with the post, and (ii) the number of reputation units to add and subtract.

We extend the on-chain data stored per post: instead of just storing the nullifier N, we store $\{N, \bar{h}, \bar{u}\}$, where:

- $\bar{h} = hash(h, r)$ where h is the block height of the state root that was referenced in the proof
- $\bar{u} = hash(u, r)$ where u is the account's reputation score (0 for a fresh account)

r here is simply a random value, added to prevent h and u from being uncovered by brute-force search (in cryptography jargon, adding r makes the hash a *hiding commitment*).

Suppose that a post uses a root R and stores $\{N, \bar{h}, \bar{u}\}$. In the proof, it links to a previous post, with stored data $\{N_{prev}, \bar{h}_{prev}, \bar{u}_{prev}\}$. The post's proof is also required to walk over all the reputation entries that have been published between h_{prev} and h. For each nullifier

(N), the verification function would decrypt N using the user's key k, and if the decryption outputs a valid index it would apply the reputation update. If the sum of all reputation updates is δ, the proof would finally check $u = u_{prev} + \delta$.
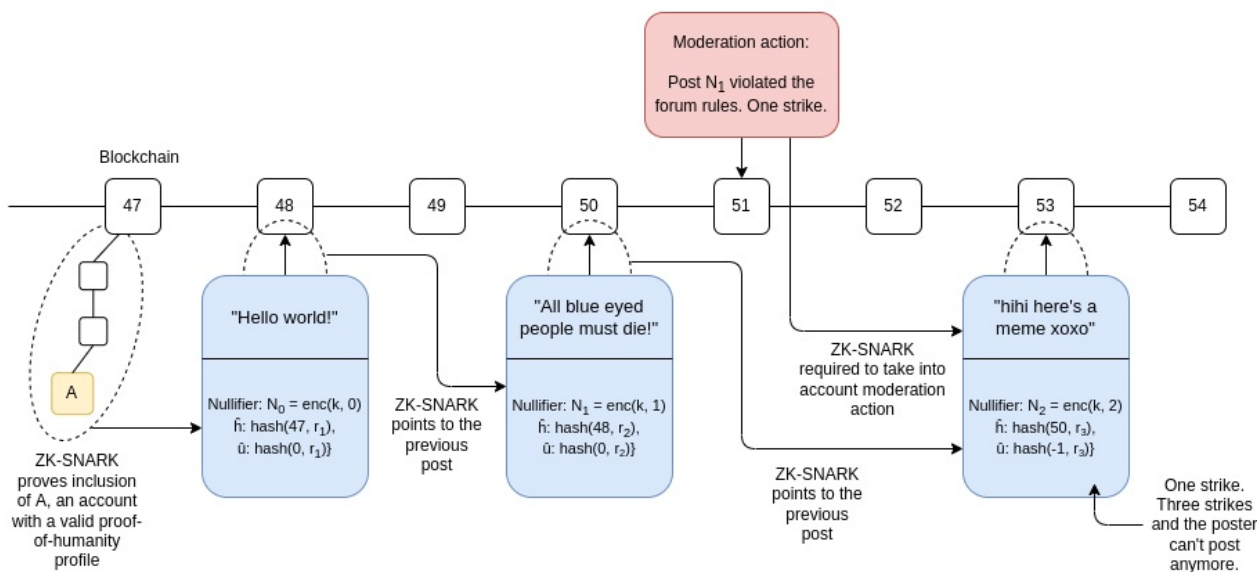
If we want a "three strikes and you're out" rule, the ZK-SNARK would also check $u > -3$. If we want a rule where a post can get a special "high-reputation poster" flag if the poster has ≥ 100 rep, we can accommodate that by adding "is $u \ge 100$?" as a public input. Many kinds of such rules can be accommodated.

To increase the scalability of the scheme, we could split it up into two kinds of messages: *posts* and *reputation update acknowledgements* (RCAs). A post would be off-chain, though it would be required to point to an RCA made in the past week. RCAs would be on-chain, and an RCA would walk through all the reputation updates since that poster's previous RCA. This way, the on-chain load is reduced to one transaction per poster per week plus one transaction per reputation message (a very low level if reputation updates are rare, eg. they're only used for moderation actions or perhaps "post of the day" style prizes).

Holding centralized parties accountable

Sometimes, you need to build a scheme that has a central "operator" of some kind. This could be for many reasons: sometimes it's for scalability, and sometimes it's for privacy - specifically, the privacy of data held by the operator.

The [MACI](#) coercion-resistant voting system, for example, requires voters to submit their votes on-chain encrypted to a secret key held by a central operator. The operator would decrypt all the votes on-chain, count them up, and reveal the final result, along with a ZK-SNARK proving that they did everything correctly. This extra complexity is necessary to ensure a strong privacy property (called **coercion-resistance**): that users cannot prove to others how they voted *even if they wanted to*.

Thanks to blockchains and ZK-SNARKs, the amount of trust in the operator can be kept very low. A malicious operator could still break coercion resistance, but because votes are published on the blockchain, the operator cannot cheat by censoring votes, and because the operator must provide a ZK-SNARK, they cannot cheat by mis-calculating the result.

Combining ZK-SNARKs with MPC

A more advanced use of ZK-SNARKs involves making proofs over computations where the inputs are split between two or more parties, and we don't want each party to learn the other parties' inputs. You can satisfy the privacy requirement with [garbled circuits](#) in the 2-party case, and more complicated multi-party computation protocols in the N-party case. ZK-SNARKs can be combined with these protocols to do verifiable multi-party computation.

This could enable more advanced reputation systems where multiple participants can perform joint computations over their private inputs, it could enable privacy-preserving but authenticated data markets, and many other applications. That said, note that the math for doing this efficiently is still relatively in its infancy.

What can't we make private?

ZK-SNARKs are generally very effective for creating systems where users have private state. **But ZK-SNARKs cannot hold private state that *nobody* knows.** To make a proof about a piece of information, the prover has to know that piece of information in cleartext.

A simple example of what can't (easily) be made private is Uniswap. In Uniswap, there is a single [logically-central](#) "thing", the market maker account, which belongs to no one, and every single trade on Uniswap is trading against the market maker account. You can't hide the state of the market maker account, because then someone would have to hold the state in cleartext to make proofs, and their active involvement would be necessary in every single transaction.

You *could* make a centrally-operated, but safe and private, Uniswap with ZK-SNARKed garbled circuits, but it's not clear that the benefits of doing this are worth the costs. There may not even be any real benefit: the contract would need to be able to tell users what the prices of the assets are, and the block-by-block changes in the prices tell a lot about what the trading activity is.

Blockchains can make state information *global*, ZK-SNARKs can make state information *private*, but we don't really have any good way to make state information *global and private* at the same time.

Edit: you can use multi-party computation to implement shared private state. But this requires an honest-majority threshold assumption, and one that's likely unstable in practice because (unlike eg. with 51% attacks) a malicious majority could collude to break the privacy without ever being detected.

Putting the primitives together

In the sections above, we've seen some examples that are powerful and useful tools by themselves, but they are also intended to serve as building blocks in other applications. Nullifiers, for example, are important for currency, but it turns out that they pop up again and again in all kinds of use cases.

The "forced chaining" technique used in the negative reputation section is very broadly applicable. It's effective for many applications where users have complex "profiles" that change in complex ways over time, and you want to force the users to follow the rules of the system while preserving privacy so no one sees which user is performing which action. Users could even be required to have entire private Merkle trees representing their internal "state". The "commitment pool" gadget [proposed in this post](#) could be built with ZK-SNARKs. And if some application can't be entirely on-chain and must have a centralized operator, the exact same techniques can be used to keep the operator honest too.

ZK-SNARKs are a really powerful tool for combining together the benefits of accountability and privacy. They do have their limits, though in some cases clever application design can work around those limits. I expect to see many more applications using ZK-SNARKs, and eventually applications combining ZK-SNARKs with other forms of cryptography, to be built in the years to come.

Where to use a blockchain in non-financial applications?

2022 Jun 12

Special thanks to Shrey Jain and Puja Ohlhaver for substantial feedback and review

Recently, there has been a growing amount of interest in using blockchains for not-just-financial applications. This is a trend that I have been strongly in favor of, for various reasons. In the last month, Puja Ohlhaver, Glen Weyl and I collaborated on a paper describing a more detailed vision for what could be done with a richer ecosystem of soulbound tokens making claims describing various kinds of relationships. This has led to some discussion, particularly focused on whether or not it makes any sense to use a blockchain in a decentralized identity ecosystem:

- Kate Sills argues for off-chain signed claims
- Puja Ohlhaver responds to Kate Sills
- Evin McMullen and myself have a podcast debating on-chain vs off-chain attestations
- Kevin Yu writes a technical overview bringing up the on-chain versus off-chain question
- Molly White argues a pessimistic case against self-sovereign identity
- Shrey Jain makes a meta-thread containing the above and many other Twitter discussions

It's worth zooming out and asking a broader question: where does it make sense, in general, to use a blockchain in non-financial applications? Should we move toward a world where even decentralized chat apps work by every message being an on-chain transaction containing the encrypted message? Or, alternatively, are blockchains only good for finance (say, because network effects mean that money has a unique need for a "global view"), with all other applications better done using centralized or more local systems?

My own view tends to be, like with blockchain voting, far from the "blockchain everywhere" viewpoint, but also far from a "blockchain minimalist". I see the value of blockchains in many situations, sometimes for *really important* goals like trust and censorship resistance but sometimes purely for convenience. This post will attempt to describe some types of situations where blockchains might be useful, especially in the context of identity, and where they are not. **This post is not a complete list and intentionally leaves many things out. The goal is rather to elucidate some common categories.**

User account key changes and recovery

One of the biggest challenges in a cryptographic account system is the issue of key changes. This can happen in a few cases:

1. You're worried that your current key might get lost or stolen, and you want to **switch to a different key**
2. You want to **switch to a different cryptographic algorithm** (eg. because you're worried quantum computers will come soon and you want to upgrade to post-quantum)
3. Your **key got lost**, and you want to regain access to your account
4. Your **key got stolen**, and you want to regain exclusive access to your account (and you don't want the thief to be able to do the same)

[1] and [2] are relatively simple in that they can be done in a fully *self-sovereign* way: you control key X, you want to switch to key Y, so you publish a message signed with X saying "Authenticate me with Y from now on", and everyone accepts that.

But notice that **even for these simpler key change scenarios, you can't *just* use cryptography**. Consider the following sequence of events:

- You are worried that key A might get stolen, so you sign a message with A saying "I use B now"
- A year later, a hacker actually does steal key A. They sign a message saying with A saying "I use C now", where C is their own key

From the point of view of someone coming in later who just receives these two messages, they see that A is no longer used, but they don't know whether "replace A with B" or "replace A with C" has higher priority.
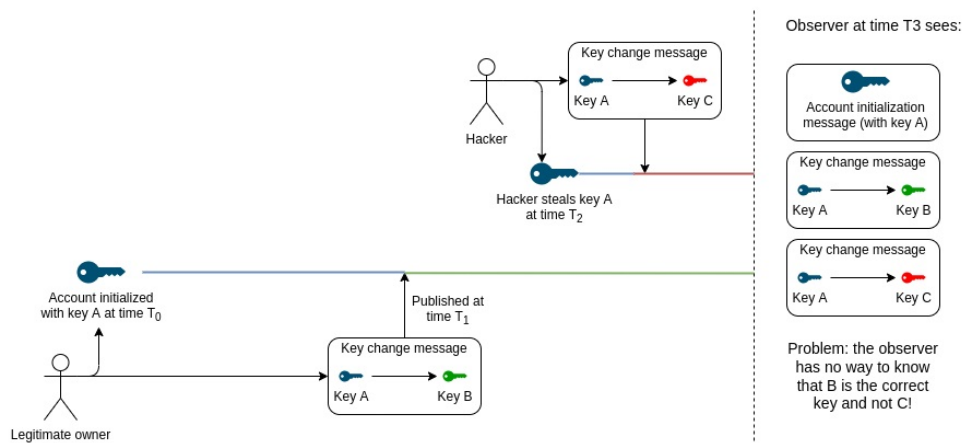
This is equivalent to the famous double-spend problem in designing decentralized currencies, except instead of the goal being to prevent a previous owner of a coin from being able to send it again, here the goal is to prevent the previous key controlling an account from being able to change the key. **Just like creating a decentralized currency, doing account management in a decentralized way requires something like a blockchain**. A blockchain can timestamp the key change messages, providing common knowledge over whether B or C came first.

[3] and [4] are harder. In general, my own preferred solution is multisig and social recovery wallets, where a group of friends, family members and other contacts can transfer control of your account to a new key if it gets lost or stolen. For critical operations (eg. transferring large quantities of funds, or signing an important contract), participation of this group can also be required.

But this too requires a blockchain. Social recovery using secret sharing is possible, but it is more difficult in practice: if you no longer trust some of your contacts, or if they want to change their own keys, you have no way to revoke access without changing your key yourself. And so we're back to requiring some form of on-chain record.

One subtle but important idea in [the DeSoc paper](#) is that to preserve non-transferability, social recovery (or "community recovery") of profiles might actually need to be *mandatory*. That is, even if you sell your account, you can always use community recovery to get the account back. This would solve problems like not-actually-reputable drivers [buying verified accounts](#) on ride sharing platforms. That said, this is a speculative idea and does not have to be fully implemented to get the other benefits of blockchain-based identity and reputation systems.

Note that so far this is a limited use-case of blockchains: it's totally okay to have accounts on-chain but do everything else off-chain. There's a place for these kinds of hybrid visions; [Sign-in With Ethereum](#) is good simple example of how this could be done in practice.

Modifying and revoking attestations

Alice goes to Example College and gets a degree in example studies. She gets a digital record certifying this, signed with Example College's keys. Unfortunately, six months later, Example College discovers that Alice had committed a large amount of plagiarism, and revokes her degree. But Alice continues to use her old digital record to go around claiming to various people and institutions that she has a degree. Potentially, the attestation could even carry *permissions* - for example, the right to log in to the college's online forum - and Alice might try to inappropriately access that too. How do we prevent this?

The "blockchain maximalist" approach would be to make the degree an on-chain NFT, so Example College can then issue an on-chain transaction to revoke the NFT. But perhaps this is needlessly expensive: issuance is common, revocation is rare, and we don't want to require Example College to issue transactions and pay fees for every issuance if they don't have to. So instead we can go with a hybrid solution: **make initial degree an off-chain signed message, and do revocations on-chain**. This is the [approach that OpenCerts uses](#).

The *fully off-chain* solution, and the one advocated by [many off-chain verifiable credentials proponents](#), is that Example College runs a server where they publish a full list of their revocations (to improve privacy, each attestation can come with an attached nonce and the revocation list can just be a list of nonces).

For a college, running a server is not a large burden. But for any smaller organization or individual, managing "yet another server script" and making sure it stays online is a significant burden for IT people. **If we tell people to "just use a server" out of blockchain-phobia, then the likely outcome is that everyone outsources the task to a centralized provider.** Better to keep the system decentralized and just use a blockchain - especially now that rollups, sharding and other techniques are finally starting to come online to make the cost of a blockchain cheaper and cheaper.

Negative reputation

Another important area where off-chain signatures do not suffice is **negative reputation** - that is, attestations where the person or organization that you're making attestations about might not want you to see them. I'm using "negative reputation" here as a technical term: the most obvious motivating use case is attestations saying bad things about someone, like a bad review or a report that someone acted abusively in some context, but there are also use cases where "negative" attestations don't imply bad behavior - for example, taking out a loan and wanting to prove that you have not taken out too many other loans at the same time.

With off-chain claims, you can do *positive* reputation, because it's in the interest of the recipient of a claim to show it to appear more reputable (or make a ZK-proof about it), but you can't do *negative* reputation, because someone can always choose to only show the claims that make them look good and leave out all the others.

Here, making attestations on-chain actually does fix things. To protect privacy, we can add encryption and zero knowledge proofs: an attestation can just be an on-chain record with data encrypted to the recipient's public key, and users could prove lack of negative reputation by running a zero knowledge proof that walks over the entire history of records on chain. The proofs being on-chain and the verification process being blockchain-aware makes it easy to verify that the proof actually did walk over the *whole* history and did not skip any records. To make this computationally feasible, a user could use [incrementally verifiable computation](#) (eg. [Halo](#)) to maintain and prove a tree of records that were encrypted to them, and then reveal parts of the tree when needed.

Negative reputation and revoking attestations are in some sense equivalent problems: you can revoke an attestation by adding another negative-reputation attestation saying "this other attestation doesn't count anymore", and you can implement negative reputation with revocation by piggybacking on positive reputation: Alice's degree at Example College could be revoked and replaced with a degree saying "Alice got a degree in example studies, but she took out a loan".

Is negative reputation a good idea?

One critique of negative reputation that we sometimes hear is: but isn't negative reputation a dystopian scheme of "*scarlet letters*", and shouldn't we try our best to do things with positive reputation instead?

Here, while I support the goal of avoiding *unlimited* negative reputation, I disagree with the idea of avoiding it entirely. Negative reputation is important for many use cases. Uncollateralized lending, which is highly valuable for improving capital efficiency within the blockchain space and outside, clearly benefits from it. [Unirep Social](#) shows a proof-of-concept social media platform that combines a high level of anonymity with a privacy-preserving negative reputation system to limit abuse.

Sometimes, negative reputation can be empowering and positive reputation can be exclusionary. An online forum where [every unique human](#) gets the right to post until they get too many "strikes" for misbehavior is more egalitarian than a forum that requires some kind of "proof of good character" to be admitted and allowed to speak in the first place. Marginalized people whose lives are mostly "outside the system", even if they actually are of good character, would have a hard time getting such proofs.

Readers of the strong civil-libertarian persuasion may also want to consider the case of an anonymous reputation system for clients of sex workers: you want to protect privacy, but you also might want a system where if a client mistreats a sex worker, they get a "black mark" that encourages other workers to be more careful or stay away. In this way, negative reputation that's hard to hide can actually empower the vulnerable and protect safety. The point here is not to defend some *specific* scheme for negative reputation; rather, it's to show that there's very real value that negative reputation unlocks, and a successful system needs to support it *somehow*.

Negative reputation does not have to be *unlimited* negative reputation: I would argue that it should always be possible to create a new profile at some cost (perhaps sacrificing a lot or all of your existing positive reputation). There is a balance between [too little accountability and too much accountability](#). But having *some* technology that makes negative reputation possible in the first place is a prerequisite for unlocking this design space.

Committing to scarcity

Another example of where blockchains are valuable is issuing attestations that have a provably limited quantity. If I want to make an endorsement for someone (eg. one might imagine a company looking for jobs or a government visa program looking at such endorsements), the third party looking at the endorsement would want to know whether I'm careful with endorsements or if I give them off to pretty much whoever is friends with me and asks nicely.

The ideal solution to this problem would be to make endorsements public, so that endorsements become incentive-aligned: if I endorse someone who turns out to do something wrong, everyone can discount my endorsements in the future. But often, we also want to preserve privacy. So instead what I could do is publish hashes of each endorsement on-chain, so that anyone can see how many I have given out.

An even more effective usecase is many-at-a-time issuance: if an artists wants to issue N copies of a "limited-edition" NFT, they could publish

on-chain a *single* hash containing the Merkle root of the NFTs that they are issuing. The single issuance prevents them from issuing more after the fact, and you can publish the number (eg. 100) signifying the quantity limit along with the Merkle root, signifying that only the leftmost 100 Merkle branches are valid.
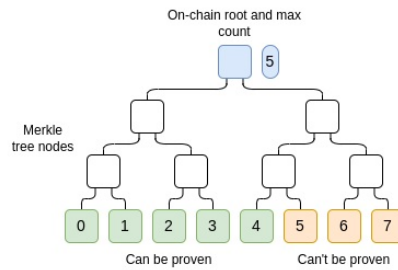
By publishing a single Merkle root and max count on-chain, you can commit issue a limited quantity of attestations. In this example, there are only five possible valid Merkle branches that could satisfy the proof check. Astute readers may notice a conceptual similarity to Plasma chains.

Common knowledge

One of the powerful properties of blockchains is that they create *common knowledge*: if I publish something on-chain, then Alice can see it, Alice can see that Bob can see it, Charlie can see that Alice can see that Bob can see it, and so on.

Common knowledge is often important for coordination. For example, a group of people might want to speak out about an issue, but only feel comfortable doing so if there's enough of them speaking out at the same time that they have safety in numbers. One possible way to do this is for one person to start a "commitment pool" around a particular statement, and invite others to publish hashes (which are private at first) denoting their agreement. Only if enough people participate within some period of time, all participants would be required to have their next on-chain message publicly reveal their position.

A design like this could be accomplished with a combination of zero knowledge proofs and blockchains (it could be done without blockchains, but that requires either witness encryption, which is not yet available, or trusted hardware, which has deeply problematic security assumptions). There is a large design space around these kinds of ideas that is very underexplored today, but could easily start to grow once the ecosystem around blockchains and cryptographic tools grows further.

Interoperability with other blockchain applications

This is an easy one: some things should be on-chain to better interoperate with other on-chain applications. Proof of humanity being an on-chain NFT makes it easier for projects to automatically airdrop or give governance rights to accounts that have proof of humanity profiles. Oracle data being on-chain makes it easier for defi projects to read. In all of these cases, the blockchain does not remove the need for trust, though it can house structures like DAOs that manage the trust. But the main value that being on-chain provides is simply *being in the same place as the stuff that you're interacting with, which needs a blockchain for other reasons*.

Sure, you *could* run an oracle off-chain and require the data to be imported only when it needs to be read, but in many cases that would actually be *more* expensive, and needlessly impose complexity and costs on developers.

Open-source metrics

One key goal of the Decentralized Society paper is the idea that it should be possible to *make calculations* over the graph of attestations. A really important one is **measuring decentralization** and diversity. For example, many people seem to agree that an ideal voting mechanism would somehow keep diversity in mind, giving greater weight to projects that are supported not just by the largest number of *coins* or even *humans*, but by the largest number of *truly distinct perspectives*.

Quadratic funding as implemented in Gitcoin Grants also includes some explicitly diversity-favoring logic to mitigate attacks.

Another natural place where measurements and scores are going to be valuable is **reputation systems**. This already exists in a centralized form with ratings, but it can be done in a much more decentralized way where the algorithm is transparent while at the same time preserving more user privacy.

Aside from tightly-coupled use cases like this, where attempts to measure to what extent some set of people is connected and feed that directly into a mechanism, there's also broader use case of helping a community understand itself. In the case of measuring decentralization, this might be a matter of identifying areas where concentration is getting too high, which might require a response. In all of these cases, running computerized algorithms over large bodies of attestations and commitments and doing actually important things with the outputs is going to be unavoidable.

We should not try to *abolish* quantified metrics, we should try to make *better* ones

Kate Sills expressed her skepticism of the goal of making calculations over reputation, an argument that applies both for public analytics and for individuals ZK-proving over their reputation (as in Unirep Social):

> The process of evaluating a claim is very subjective and context-dependent. People will naturally disagree about the trustworthiness of other people, and trust depends on the context ... [because of this] we should be extremely skeptical of any proposal to "calculate over" claims to get objective results.

I this case, I agree with the importance of subjectivity and context, but I would disagree with the more expansive claim that avoiding calculations around reputation entirely is the right goal to be aiming towards. Pure individualized analysis does not scale far beyond Dunbar's number, and any complex society that is attempting to support large-scale cooperation has to rely on aggregations and simplifications to some extent.

That said, I would argue that an open-participation ecosystem of attestations (as opposed to the centralized one we have today) can get us the best of both worlds by opening up space for *better* metrics. Here are some principles that such designs could follow:

- **Inter-subjectivity**: eg. a reputation should not be a single global score; instead, it should be a more subjective calculation involving the person or entity being evaluated but also the viewer checking the score, and potentially even other aspects of the local context.
- **Credible neutrality**: the scheme should clearly not leave room for powerful elites to constantly manipulate it in their own favor. Some possible ways to achieve this are maximum **transparency** and **infrequent change** of the algorithm.
- **Openness**: the ability to make meaningful inputs, and to audit other people's outputs by running the check yourself, should be open to anyone, and not just restricted to a small number of powerful groups.

If we don't create good large-scale aggregates of social data, then we risk ceding market share to opaque and centralized social credit scores instead.

Not all data should be on-chain, but making *some* data public in a common-knowledge way can help increase a community's legibility to itself without creating data-access disparities that could be abused to centralize control.

As a data store

This is the really controversial use case, even among those who accept most of the others. There is a common viewpoint in the blockchain space that blockchains should only be used in those cases where they are truly needed and unavoidable, and everywhere else we should use other tools.

This attitude makes sense in a world where transaction fees are very expensive, and blockchains are uniquely incredibly inefficient. But it makes less sense in a world where blockchains have rollups and sharding and transaction fees have dropped down to a few cents, and the difference in redundancy between a blockchain and non-blockchain decentralized storage might only be 100x.

Even in such a world, it would not make sense to store *all* data on-chain. But small text records? Absolutely. Why? Because **blockchains are just a really convenient place to store stuff**. I maintain a copy of this blog on IPFS. But uploading to IPFS often takes an hour, it requires centralized gateways for users to access it with anything close to website levels of latency, and occasionally files drop off and no longer become visible. Dumping the entire blog on-chain, on the other hand, would solve that problem completely. Of course, the blog is too big to *actually* be dumped on-chain, even post-sharding, but the same principle applies to smaller records.

Some examples of small cases where putting data on-chain just to store it may be the right decision include:

- **Augmented secret sharing**: splitting your password into `N` pieces where any `M = N-R` of the pieces can recover the password, but in a way where you can choose the contents of all `N` of the pieces. For example, the pieces could all be hashes of passwords, secrets generated through some other tool, or answers to security questions. This is done by publishing an extra `R` pieces (which are random-looking) on-chain, and doing `N-of-(N+R)` secret sharing on the whole set.
- **ENS optimization**. ENS *could* be made more efficient by combining all records into a single hash, only publishing the hash on-chain, and requiring anyone accessing the data to get the full data off of IPFS. But this would significantly increase complexity, and add yet another software dependency. And so ENS keeps data on-chain even if it is longer than 32 bytes.
- **Social metadata** - data connected to your account (eg. for sign-in-with-Ethereum purposes) that you want to be public and that is very short in length. This is generally not true for larger data like profile pictures (though if the picture happens to be a small SVG file it could be!), but it is true for text records.
- **Attestations and access permissions**. Especially if the data being stored is less than a few hundred bytes long, it might be more convenient to store the data on-chain than put the hash on-chain and the data off-chain.

In a lot of these cases, the tradeoff isn't just cost but also privacy in those edge cases where keys or cryptography break. Sometimes, privacy is only somewhat important, and the occasional loss of privacy from leaked keys or the faraway specter of quantum computing revealing everything in 30 years is less important than having a very high degree of certainty that the data will remain accessible. After all, off-chain data stored in your "data wallet" can get hacked too.

But sometimes, data is particularly sensitive, and that can be another argument against putting it on-chain, and keeping it stored locally as a second layer of defense. But note that in those cases, that privacy need is an argument not just against blockchains, but against *all* decentralized storage.

Conclusions

Out of the above list, the two I am personally by far the most confident about are **interoperability with other blockchain applications** and **account management**. The first is on-chain already, and the second is relatively cheap (need to use the chain once per user, and not once per action), the case for it is clear, and there really isn't a good non-blockchain-based solution.

Negative reputation and **revocations** are also important, though they are still relatively early-stage use cases. A lot can be done with reputation by relying on off-chain positive reputation only, but I expect that the case for revocation and negative reputation will become more clear over time. I expect there to be attempts to do it with centralized servers, but over time it should become clear that blockchains are the only way to avoid a hard choice between inconvenience and centralization.

Blockchains as data stores for short text records may be marginal or may be significant, but I do expect at least some of that kind of usage to keep happening. Blockchains really are just incredibly convenient for cheap and reliable data retrieval, where data continues to be retrievable whether the application has two users or two million. **Open-source metrics** are still a very early-stage idea, and it remains to see just how much can be done and made open without it becoming exploitable (as eg. online reviews, social media karma and the like get exploited all the time). And **common knowledge games** require convincing people to accept entirely new workflows for socially important things, so of course that is an early-stage idea too.

I have a large degree of uncertainty in exactly what level of non-financial blockchain usage in each of these categories makes sense, but it seems clear that blockchains as an enabling tool in these areas should not be dismissed.

Two thought experiments to evaluate automated stablecoins

2022 May 25

Special thanks to Dan Robinson, Hayden Adams and Dankrad Feist for feedback and review.

The recent LUNA crash, which led to tens of billions of dollars of losses, has led to a storm of criticism of "algorithmic stablecoins" as a category, with many considering them to be a "fundamentally flawed product". The greater level of scrutiny on defi financial mechanisms, especially those that try very hard to optimize for "capital efficiency", is highly welcome. The greater acknowledgement that present performance is no guarantee of future returns (or even future lack-of-total-collapse) is even more welcome. Where the sentiment goes very wrong, however, is in painting all automated pure-crypto stablecoins with the same brush, and dismissing the entire category.

While there are plenty of automated stablecoin designs that are fundamentally flawed and doomed to collapse eventually, and plenty more that can survive theoretically but are highly risky, there are also many stablecoins that are highly robust in theory, and have survived extreme tests of crypto market conditions in practice. Hence, what we need is not stablecoin boosterism *or* stablecoin doomerism, but rather a return to *principles-based thinking*. So what *are* some good principles for evaluating whether or not a particular automated stablecoin is a truly stable one? For me, the test that I start from is asking how the stablecoin responds to two thought experiments.

Click here to skip straight to the thought experiments.

Reminder: what is an automated stablecoin?

For the purposes of this post, an automated stablecoin is a system that has the following properties:

1. It issues a **stablecoin**, which attempts to **target** a particular **price index**. Usually, the target is 1 USD, but there are other options too. There is some **targeting mechanism** that continuously works to push the price toward the index if it veers away in either direction. This makes ETH and BTC not stablecoins (duh).
2. The targeting mechanism is **completely decentralized**, and free of protocol-level dependencies on specific trusted actors. Particularly, it must **not rely on asset custodians**. This makes USDT and USDC not automated stablecoins.

In practice, (2) means that the targeting mechanism must be some kind of smart contract which manages some reserve of crypto-assets, and uses those crypto-assets to prop up the price if it drops.

How does Terra work?

Terra-style stablecoins (roughly the same family as seignorage shares, though many implementation details differ) work by having a pair of two coins, which we'll call a *stablecoin* and a *volatile-coin* or *volcoin* (in Terra, UST is the stablecoin and LUNA is the volcoin). The stablecoin retains stability using a simple mechanism:

- If the price of the stablecoin exceeds the target, the system auctions off new stablecoins (and uses the revenue to burn volcoins) until the price returns to the target
- If the price of the stablecoin drops below the target, the system buys back and burns stablecoins (issuing new volcoins to fund the burn) until the price returns to the target
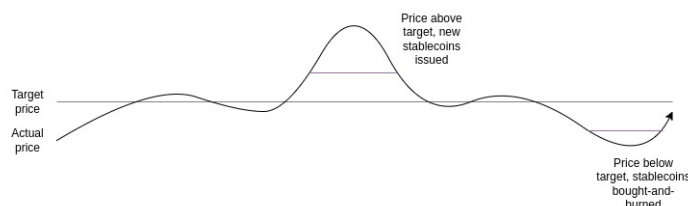
Now what is the price of the volcoin? The volcoin's value could be purely speculative, backed by an assumption of greater stablecoin demand in the future (which would require burning volcoins to issue). Alternatively, the value could come from fees: either trading fees on stablecoin <-> volcoin exchange, or holding fees charged per year to stablecoin holders, or both. But in all cases, **the price of the volcoin comes from the expectation of future activity in the system**.

How does RAI work?

In this post I'm focusing on RAI rather than DAI because RAI better exemplifies the pure "ideal type" of a collateralized automated stablecoin, backed by ETH only. DAI is a hybrid system backed by both centralized and decentralized collateral, which is a reasonable choice for their product but it does make analysis trickier.

In RAI, there are two main categories of participants (there's also holders of FLX, the speculative token, but they play a less important role):

- A **RAI holder** holds RAI, the stablecoin of the RAI system.
- A **RAI lender** deposits some ETH into a smart contract object called a "**safe**". They can then withdraw RAI up to the value of $\frac{2}{3}$ of that ETH (eg. if 1 ETH = 100 RAI, then if you deposit 10 ETH you can withdraw up to $10 * 100 * \frac{2}{3} \approx 667$ RAI). A lender can recover the ETH in the same if they pay back their RAI debt.

There are two main reasons to become a RAI lender:

1. **To go long on ETH**: if you deposit 10 ETH and withdraw 500 RAI in the above example, you end up with a position worth 500 RAI but with 10 ETH of exposure, so it goes up/down by 2% for every 1% change in the ETH price.
2. **Arbitrage** if you find a fiat-denominated investment that goes up faster than RAI, you can borrow RAI, put the funds into that investment, and earn a profit on the difference.

If the ETH price drops, and a safe no longer has enough collateral (meaning, the RAI debt is now more than $\frac{2}{3}$ times the value of the ETH deposited), a **liquidation** event takes place. The safe gets auctioned off for anyone else to buy by putting up more collateral.

The other main mechanism to understand is **redemption rate adjustment**. In RAI, the target isn't a fixed quantity of USD; instead, it moves up or down, and the rate at which it moves up or down adjusts in response to market conditions:

- **If the price of RAI is above the target, the redemption rate decreases**, reducing the incentive to hold RAI and increasing the incentive to hold negative RAI by being a lender. This pushes the price back down.
- **If the price of RAI is below the target, the redemption rate increases**, increasing the incentive to hold RAI and reducing the incentive to hold negative RAI by being a lender. This pushes the price back up.
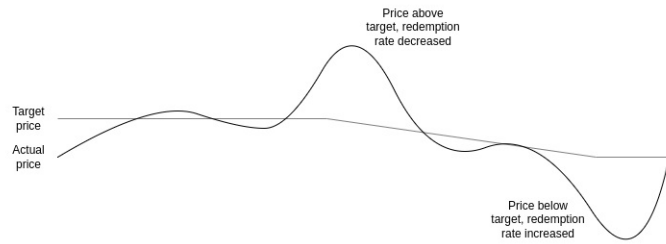
Thought experiment 1: can the stablecoin, even in theory, safely "wind down" to zero users?

In the non-crypto real world, nothing lasts forever. Companies shut down all the time, either because they never manage to find enough users in the first place, or because once-strong demand for their product is no longer there, or because they get displaced by a superior competitor. Sometimes, there are partial collapses, declines from mainstream status to niche status (eg. MySpace). Such things have to happen to make room for new products. But in the non-crypto world, when a product shuts down or declines, *customers* generally don't get hurt all that much. There are certainly some cases of people falling through the cracks, but on the whole shutdowns are orderly and the problem is manageable.

But what about automated stablecoins? What happens if we look at a stablecoin from the bold and radical perspective that the system's ability to avoid collapsing and losing huge amounts of user funds should *not* depend on a constant influx of new users? Let's see and find out!

Can Terra wind down?

In Terra, the price of the volcoin (LUNA) comes from the expectation of fees from future activity in the system. **So what happens if expected future activity drops to near-zero**? The market cap of the volcoin drops until it becomes quite small relative to the stablecoin. At that point, **the system becomes extremely fragile**: only a small downward shock to demand for the stablecoin could lead to the targeting mechanism printing lots of volcoins, which causes the volcoin to hyperinflate, at which point the stablecoin too loses its value.

The system's collapse can even become a self-fulfilling prophecy: if it seems like a collapse is likely, this reduces the expectation of future fees that is the basis of the value of the volcoin, pushing the volcoin's market cap down, making the system even more fragile and potentially triggering that very collapse - exactly as we saw happen with Terra in May.

First, the volcoin price drops. Then, the stablecoin starts to shake. The system attempts to shore up stablecoin demand by issuing more volcoins. With confidence in the system low, there are few buyers, so the volcoin price rapidly falls. Finally, once the volcoin price is near-zero, the stablecoin too collapses.

In principle, if demand decreases extremely slowly, the volcoin's expected future fees and hence its market cap could still be large relative to the stablecoin, and so the system would continue to be stable at every step of its decline. **But this kind of successful slowly-decreasing managed decline is very unlikely. What's more likely is a rapid drop in interest followed by a bang.**
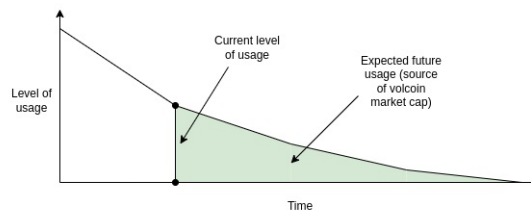
Safe wind-down: at every step, there's enough expected future revenue to justify enough volcoin market cap to keep the stablecoin safe at its current level.
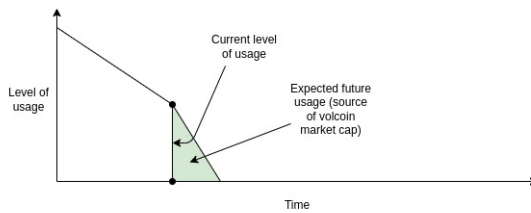
Unsafe wind-down: at some point, there's not enough expected future revenue to justify enough volcoin market cap to keep the stablecoin safe. Collapse is likely.

Can RAI wind down?

RAI's security depends on an asset external to the RAI system (ETH), so RAI has a much easier time safely winding down. If the decline in demand is unbalanced (so, either demand for holding drops faster or demand for lending drops faster), the redemption rate will adjust to equalize the two. The lenders are holding a leveraged position in ETH, not FLX, so there's no risk of a positive-feedback loop where reduced confidence in RAI causes demand for lending to also decrease.

If, in the extreme case, all demand for holding RAI disappears simultaneously except for one holder, the redemption rate would skyrocket until eventually every lender's safe gets liquidated. The single remaining holder would be able to buy the safe in the liquidation auction, use their RAI to immediately clear its debt, and withdraw the ETH. This gives them the opportunity to get a fair price for their RAI, paid for out of the ETH in the safe.

Another extreme case worth examining is where RAI becomes the *primary* appliation on Ethereum. In this case, a reduction in expected future demand for RAI would crater the price of ETH. In the extreme case, a cascade of liquidations is possible, leading to a messy collapse of the system. But RAI is far more robust against this possibility than a Terra-style system.

Thought experiment 2: what happens if you try to peg the stablecoin to an index that goes up 20% per year?

Currently, stablecoins tend to be pegged to the US dollar. RAI stands out as a slight exception, because its peg adjusts up or down due to the redemption rate and the peg started at 3.14 USD instead of 1 USD (the exact starting value was a concession to being normie-friendly, as a true math nerd would have chosen tau = 6.28 USD instead). But they do not have to be. You can have a stablecoin pegged to a basket of assets, a consumer price index, or some arbitrarily complex formula ("a quantity of value sufficient to buy {global average CO2 concentration minus 375} hectares of land in the forests of Yakutia"). As long as you can find an oracle to prove the index, and people to participate on all sides of the market, you can make such a stablecoin work.

As a thought experiment to evaluate sustainability, let's imagine a stablecoin with a particular index: a quantity of US dollars that grows by 20% per year. In math language, the index is $1.2^{(t - t_0)}$ USD, where t is the current time in years and t_0 is the time when the system launched. An even more fun alternative is $1.04^{\frac{1}{2}*(t - t_0)^2}$ USD, so it starts off acting like a regular USD-denominated stablecoin, but the USD-denominated return rate keeps increasing by 4% every year.
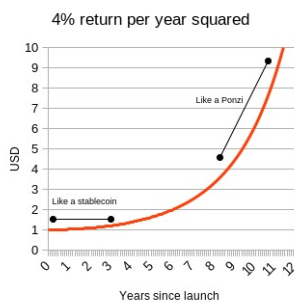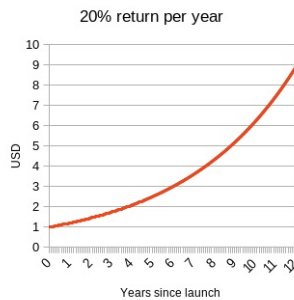
Obviously, there is no genuine investment that can get anywhere close to 20% returns per year, and there is *definitely* no genuine investment that can keep increasing its return rate by 4% per year forever. *But what happens if you try*?

I will claim that there's *basically* two ways for a stablecoin that tries to track such an index to turn out:

1. It charges some kind of negative interest rate on holders that equilibrates to basically cancel out the USD-denominated growth rate built in to the index.
2. It turns into a Ponzi, giving stablecoin holders amazing returns for some time until one day it suddenly collapses with a bang.

It should be pretty easy to understand why RAI does (1) and LUNA does (2), and so RAI is better than LUNA. But this also shows a deeper and more important fact about stablecoins: **for a collateralized automated stablecoin to be sustainable, it has to *somehow* contain the possibility of implementing a negative interest rate.** A version of RAI programmatically prevented from implementing negative interest rates (which is what the earlier single-collateral DAI basically was) would also turn into a Ponzi if tethered to a rapidly-appreciating price index.

Even outside of crazy hypotheticals where you build a stablecoin to track a Ponzi index, the stablecoin must *somehow* be able to respond to situations where even at a zero interest rate, demand for holding exceeds demand for borrowing. If you don't, the price rises above the peg, and the stablecoin becomes vulnerable to price movements in both directions that are quite unpredictable.

Negative interest rates can be done in two ways:

1. RAI-style, having a floating target that can drop over time if the redemption rate is negative

2. Actually having balances decrease over time

Option (1) has the user-experience flaw that the stablecoin no longer cleanly tracks "1 USD". Option (2) has the developer-experience flaw that developers aren't used to dealing with assets where receiving N coins does not unconditionally mean that you can later send N coins. But choosing one of the two seems unavoidable - unless you go the MakerDAO route of being a *hybrid* stablecoin that uses both pure cryptoassets and centralized assets like USDC as collateral.

What can we learn?

In general, the crypto space needs to move away from the attitude that it's okay to achieve safety by relying on endless growth. It's certainly not acceptable to maintain that attitude by saying that "the fiat world works in the same way", because the fiat world is not attempting to offer anyone returns that go up much faster than the regular economy, outside of isolated cases that certainly should be criticized with the same ferocity.

Instead, while we certainly should hope for growth, we should evaluate how safe systems are by looking at their steady state, and even the pessimistic state of how they would fare under extreme conditions and ultimately whether or not they can safely wind down. If a system passes this test, that does not mean it's safe; it could still be fragile for other reasons (eg. insufficient collateral ratios), or have bugs or governance vulnerabilities. But steady-state and extreme-case soundness should always be one of the first things that we check for.

In Defense of Bitcoin Maximalism

2022 Apr 01

We've been hearing for years that the future is [blockchain, not Bitcoin](). The future of the world won't be one major cryptocurrency, or even a few, but many cryptocurrencies - and the winning ones will have strong leadership under one central roof to adapt rapidly to users' needs for scale. Bitcoin is a [boomer]() [coin](), and Ethereum is soon to follow; it will be newer and more energetic assets that attract the new waves of mass users who don't care about weird libertarian ideology or "self-sovereign verification", are turned off by toxicity and anti-government mentality, and just want blockchain defi and games that are *fast* and *work*.

But what if this narrative is all wrong, and the ideas, habits and practices of Bitcoin maximalism are in fact pretty close to correct? What if Bitcoin is far more than an outdated pet rock tied to a network effect? What if Bitcoin maximalists actually deeply understand that they are operating in a very hostile and uncertain world where there are things that need to be fought for, and their actions, personalities and opinions on protocol design deeply reflect that fact? What if we live in a world of *honest* cryptocurrencies (of which there are very few) and *grifter* cryptocurrencies (of which there are very many), and a healthy dose of *intolerance* is in fact necessary to prevent the former from sliding into the latter? That is the argument that this post will make.

We live in a dangerous world, and protecting freedom is serious business

Hopefully, this is much more obvious now than it was six weeks ago, when many people still seriously thought that Vladimir Putin is a misunderstood and kindly character who is merely trying to protect Russia and save Western Civilization from the gaypocalypse. But it's still worth repeating. **We live in a dangerous world, where there are plenty of bad-faith actors who do not listen to compassion and reason.**

A blockchain is at its core a security technology - a technology that is fundamentally all about protecting people and helping them survive in such an unfriendly world. It is, like the [Phial of Galadriel](), "a light to you in dark places, when all other lights go out". It is not a low-cost light, or a fluorescent hippie energy-efficient light, or a high-performance light. It is a light that sacrifices on *all* of those dimensions to optimize for one thing and one thing only: *to be a light that does when it needs to do when you're facing the toughest challenge of your life and there is a friggin twenty foot spider staring at you in the face.*

Source: [https://www.blackgate.com/2014/12/23/frodo-baggins-lady-galadriel-and-the-games-of-the-mighty/]()

Blockchains are being used every day by unbanked and underbanked people, by activists, by sex workers, by refugees, and by many other groups either who are uninteresting for profit-seeking centralized financial institutions to serve, or who have enemies that don't *want* them to be served. They are used as a primary lifeline by many people to make their payments and store their savings.

And to that end, public blockchains sacrifice *a lot* for security:

- Blockchains require each transaction to be independently verified thousands of times to be accepted.
- Unlike centralized systems that confirm transactions in a few hundred milliseconds, blockchains require users to wait anywhere from 10 seconds to 10 minutes to get a confirmation.
- Blockchains require users to be fully in charge of authenticating themselves: if you lose your key, you lose your coins.
- Blockchains sacrifice privacy, requiring [even crazier and more expensive technology]() to get that privacy back.

What are all of these sacrifices for? To create a system that can survive in an unfriendly world, and actually do the job of being "a light in dark places, when all other lights go out".

Excellent at that task requires two key ingredients: (i) a **robust and defensible technology stack** and (ii) a **robust and defensible culture**. The key property to have in a robust and defensible technology stack is a focus on *simplicity* and *deep mathematical purity*: a 1 MB block size, a 21 million coin limit, and a simple Nakamoto consensus proof of work mechanism that even a high school student can understand. The protocol design must be easy to justify decades and centuries down the line; *the technology and parameter choices must be a work of art.*

The second ingredient is the *culture* of uncompromising, steadfast minimalism. This must be a culture that can stand unyieldingly in defending itself against corporate and government actors trying to co-opt the ecosystem from outside, as well as bad actors *inside* the crypto space trying to exploit it for personal profit, [of]() which [there]() are [many]().

Now, what do Bitcoin and Ethereum culture actually look like? Well, let's ask Kevin Pham:

Kevin Pham
@_Kevin_Pham

I don't think the Bitcoin and Ethereum community can be anymore different.

1:58 AM · Nov 7, 2017 · Twitter for Android

Don't believe this is representative? Well, let's ask Kevin Pham again:

Kevin Pham
@_Kevin_Pham

Bitcoin and Ethereum community can't be anymore different Pt. 2

2:09 AM · Nov 7, 2017 · Twitter for Android

Now, you might say, this is just Ethereum people having fun, and at the end of the day they understand what they have to do and what they are dealing with. But do they? Let's look at the kinds of people that Vitalik Buterin, the founder of Ethereum, hangs out with:
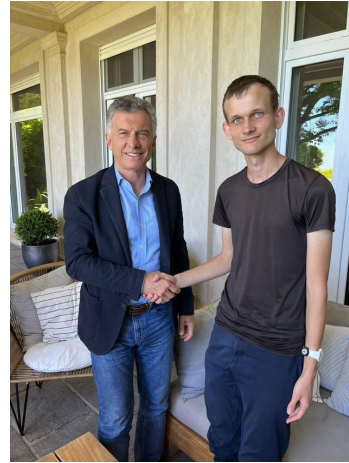

Vitalik hangs out with elite tech CEOs in Beijing, China.

Vitalik meets Vladimir Putin in Russia.

Vitalik meets Nir Bakrat, mayor of Jerusalem.

Vitalik shakes hands with Argentinian former president Mauricio Macri.

Vitalik gives a friendly hello to Eric Schmidt, former CEO of Google and advisor to US Department of Defense.

Vitalik has his first of many meetings with Audrey Tang, digital minister of Taiwan.

And this is only a small selection. The immediate question that anyone looking at this should ask is: *what the hell is the point of publicly meeting with all these people*? Some of these people are very decent entrepreneurs and politicians, but others are actively involved in serious human rights abuses that Vitalik certainly does not support. Does Vitalik not realize just how much some of these people are geopolitically at each other's throats?

Now, maybe he is just an idealistic person who believes in talking to people to help bring about world peace, and a follower of Frederick Douglass's dictum to "unite with anybody to do right and with nobody to do wrong". But there's also a simpler hypothesis: *Vitalik is a hippy-happy globetrotting pleasure and status-seeker, and he deeply enjoys meeting and feeling respected by people who are important.* And it's not just Vitalik; companies like Consensys are totally happy to partner with Saudi Arabia, and the ecosystem as a whole keeps trying to look to mainstream figures for validation.

Now ask yourself the question: when the time comes, *actually important* things are happening on the blockchain - *actually important things that offend people who are powerful* - which ecosystem would be more willing to put its foot down and refuse to censor them no matter how much pressure is applied on them to do so? The ecosystem with globe-trotting nomads who really really care about being everyone's friend, or the ecosystem with people who take pictures of themslves with an AR15 and an axe as a side hobby?

Currency is not "just the first app". It's by far the most successful one.

Many people of the "blockchain, not Bitcoin" persuasion argue that cryptocurrency is the first application of blockchains, but it's a very boring one, and the true potential of blockchains lies in bigger and more exciting things. Let's go through the list of applications in the Ethereum whitepaper:

- Issuing tokens
- Financial derivatives
- Stablecoins
- Identity and reputation systems
- Decentralized file storage
- Decentralized autonomous organizations (DAOs)
- Peer-to-peer gambling
- Prediction markets

Many of these categories have applications that have launched and that have at least *some* users. That said, cryptocurrency people really value empowering under-banked people in the "Global South". Which of these applications actually have lots of users in the Global South?

As it turns out, by far the most successful one is storing wealth and payments. 3% of Argentinians own cryptocurrency, as do 6% of Nigerians and 12% of people in Ukraine. By far the biggest instance of a government using blockchains to *accomplish something useful today* is cryptocurrency donations to the government of Ukraine, which have raised more than $100 million if you include donations to non-governmental Ukraine-related efforts.

Help Ukraine with **crypto**, don't leave us alone with the enemy

The community has already raised $ 64 172 467 / $ 200 000 000

Donate crypto to Ukraine to support people in the fight for freedom

Bitcoin (BTC)   Ethereum (ETH)
Tether (USDT ERC-20)   Tether (USDT TRC-
Terra (LUNA)   Solana (SOL)
Polkadot (DOT)   Cardano (ADA)

What other application has anywhere close to that level of actual, real adoption today? Perhaps the closest is ENS. DAOs are real and growing, but today far too many of them are appealing to wealthy rich-country people whose main interest is having fun and using cartoon-character profiles to satisfy their first-world need for self-expression, and not build schools and hospitals and solve other real world problems.

Thus, we can see the two sides pretty clearly: team "blockchain", privileged people in wealthy countries who love to virtue-signal about "moving beyond money and capitalism" and can't help being excited about "decentralized governance experimentation" as a hobby, and team "Bitcoin", a highly diverse group of both rich and poor people in many countries around the world including the Global South, who are actually using the capitalist tool of free self-sovereign money to provide real value to human beings today.

Focusing exclusively on being money makes for better money

A common misconception about why Bitcoin does not support "richly stateful" smart contracts goes as follows. Bitcoin really really values being simple, and particularly having low technical complexity, to reduce the chance that something will go wrong. As a result, it doesn't want to add the more complicated features and opcodes that are necessary to be able to support more complicated smart contracts in Ethereum.

This misconception is, of course, wrong. In fact, there are *plenty* of ways to add rich statefulness into Bitcoin; search for the word "covenants" in Bitcoin chat archives to see many proposals being discussed. And many of these proposals are surprisingly simple. The reason why covenants have not been added is *not* that Bitcoin developers see the value in rich statefulness but find even a little bit more protocol complexity intolerable. Rather, it's because Bitcoin developers are worried about the risks of the systemic complexity that rich statefulness being possible would introduce into the ecosystem!

A recent paper by Bitcoin researchers describes some ways to introduce covenants to add some degree of rich statefulness to Bitcoin.

Ethereum's battle with miner-extractable value (MEV) is an excellent example of this problem appearing in practice. It's very easy in Ethereum to build applications where the next person to interact with some contract gets a substantial reward, causing transactors and miners to fight over it, and contributing greatly to network centralization risk and requiring complicated workarounds. In Bitcoin, building such systemically risky applications is hard, in large part because Bitcoin lacks rich statefulness and focuses on the simple (and MEV-free) use case of *just being money*.

Systemic contagion can happen in non-technical ways too. Bitcoin just being money means that Bitcoin requires relatively few developers, helping to reduce the risk that developers will start demanding to print themselves free money to build new protocol features. Bitcoin just being money reduces pressure for core developers to keep adding features to "keep up with the competition" and "serve developers' needs".

In so many ways, systemic effects are real, and it's just not possible for a currency to "enable" an ecosystem of highly complex and risky decentralized applications without that complexity biting it back *somehow*. Bitcoin makes the safe choice. If Ethereum continues its layer-2-centric approach, ETH-the-currency *may* gain some distance from the application ecosystem that it's enabling and thereby get some protection. So-called high-performance layer-1 platforms, on the other hand, stand no chance.

In general, the earliest projects in an industry are the most "genuine"

Many industries and fields follow a similar pattern. First, some new exciting technology either gets invented, or gets a big leap of improvement to the point where it's actually usable for something. At the beginning, the technology is still clunky, it is too risky for almost anyone to touch as an investment, and there is no "social proof" that people can use it to become successful. As a result, the first people involved are going to be the idealists, tech geeks and others who are genuinely excited about the technology and its potential to improve society.

Once the technology proves itself enough, however, the normies come in - an event that in internet culture is often called Eternal September. And these are not just regular kindly normies who want to feel part of something exciting, but *business* normies, wearing *suits*, who start scouring the ecosystem wolf-eyed for ways to make money - with armies of venture capitalists just as eager to make their own money supporting them from the sidelines. In the extreme cases, outright *grifters* come in, creating blockchains with no redeeming social or technical value which are basically borderline scams. But the reality is that the line from "altruistic idealist" and "grifter" is really a spectrum. And the longer an ecosystem keeps going, the harder it is for any new project on the altruistic side of the spectrum to get going.

One noisy proxy for the blockchain industry's slow replacement of philosophical and idealistic values with short-term profit-seeking values is the larger and larger size of premines: the allocations that developers of a cryptocurrency give to themselves.

Major public chains: launch date vs insider allocation
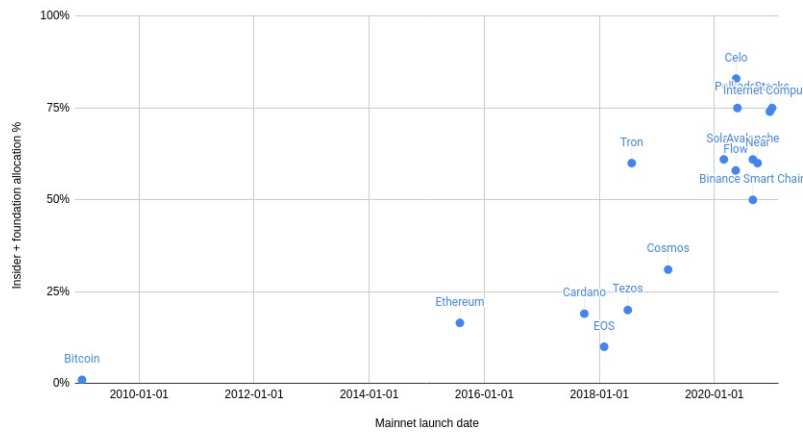
Source for insider allocations: [Messari](#).

Which blockchain communities deeply value self-sovereignty, privacy and decentralization, and are making to get big sacrifices to get it? And which blockchain communities are just trying to pump up their market caps and make money for founders and investors? The above chart should make it pretty clear.

Intolerance is good

The above makes it clear why Bitcoin's status as the first cryptocurrency gives it unique advantages that are extremely difficult for any cryptocurrency created within the last five years to replicate. But now we get to the biggest objection against Bitcoin maximalist culture: *why is it so toxic*?

The case for Bitcoin toxicity stems from [Conquest's second law](#). In Robert Conquest's original formulation, the law says that "**any organization not explicitly and constitutionally right-wing will sooner or later become left-wing**". But really, this is just a special case of a much more general pattern, and one that in the modern age of relentlessly homogenizing and conformist social media is more relevant than ever:

> If you want to retain an identity that is different from the mainstream, then you need a really strong culture that actively resists and fights
> assimilation into the mainstream every time it tries to assert its hegemony.

Blockchains are, as I mentioned above, very fundamentally and explicitly a counterculture movement that is trying to create and preserve something different from the mainstream. At a time when the world is splitting up into great power blocs that actively suppress social and economic interaction between them, blockchains are one of the very few things that can remain global. At a time when more and more people are reaching for censorship to defeat their short-term enemies, blockchains steadfastly continue to censor nothing.

The only correct way to respond to "reasonable adults" trying to tell you that to "become mainstream" you have to compromise on your "extreme" values. Because once you compromise once, you can't stop.

Blockchain communities also have to fight bad actors on the *inside*. Bad actors include:

- **Scammers**, who make and sell projects that are ultimately valueless (or worse, actively harmful) but cling to the "crypto" and "decentralization" brand (as well as highly abstract ideas of humanism and friendship) for legitimacy.
- **Collaborationists**, who publicly and loudly virtue-signal about working together with governments and actively [try](#) to [convince](#) governments to use coercive force against their competitors.
- **Corporatists**, who try to use their resources to take over the development of blockchains, and often push for protocol changes that enable centralization.

One *could* stand against all of these actors with a smiling face, politely telling the world why they "disagree with their priorities". But this is unrealistic: the bad actors will try hard to embed themselves into your community, and at that point it becomes psychologically hard to criticize them with the sufficient level of scorn that they truly require: the people you're criticizing are *friends of your friends*. And so any culture that values [agreeableness](#) will simply fold before the challenge, and let scammers roam freely through the wallets of innocent newbies.

What kind of culture *won't* fold? A culture that is willing and eager to tell both scammers on the inside and powerful opponents on the outside to [go the way of the Russian warship](#).

Weird crusades against seed oils are good

One powerful bonding tool to help a community maintain internal cohesion around its distinctive values, and avoid falling into the morass that is the mainstream, is weird beliefs and crusades that are in a similar spirit, even if not directly related, to the core mission. Ideally, these crusades should be at least partially *correct*, poking at a genuine blind spot or inconsistency of mainstream values.

The Bitcoin community is good at this. Their most recent crusade is a [war against seed oils](#), oils derived from vegetable seeds [high in omega-6](#) fatty acids that are [harmful](#) to human health.

Dietary Fat — Fatty acid content normalized to 100 percent

Oil	Saturated Fat	Oleic acid (Omega-9)	Alpha-linolenic acid (Omega-3)	Linoleic acid (Omega-6)
Canola oil	7	61	11	21
Safflower oil	8	77	1	14
Flaxseed oil	9	16	57	18
Sunflower oil	12	16	1	71
Corn oil	13	29	1	57
Olive oil	15	75	1	9
Soybean oil	15	23	8	54
Peanut oil	19	48	*	33
Cottonseed oil	27	19	*	54
Lard	43	47	1	9
Palm oil	51	39	*	10
Butterfat	68	28	1	3
Coconut oil	91	7	2	

* Trace

Saturated Fat

Monounsaturated Fat — Oleic acid (an Omega-9 fatty acid)

Polyunsaturated Fat — Alpha-linolenic acid (an Omega-3 fatty acid); Linoleic acid (an Omega-6 fatty acid)

Source: POS Pilot Plant Corporation

This Bitcoiner crusade gets treated skeptically when reviewed in the media, but the media treats the topic much more favorably when "respectable" tech firms are tackling it. The crusade helps to remind Bitcoiners that the mainstream media is fundamentally tribal and hypocritical, and so the media's shrill attempts to slander cryptocurrency as being primarily for money laundering and terrorism should be treated with the same level of scorn.

Be a maximalist

Maximalism is often derided in the media as both a dangerous toxic right-wing cult, and as a paper tiger that will disappear as soon as some other cryptocurrency comes in and takes over Bitcoin's supreme network effect. But the reality is that **none of the arguments for maximalism that I describe above depend at all on network effects**. Network effects really are logarithmic, not quadratic: once a cryptocurrency is "big enough", it has enough liquidity to function and multi-cryptocurrency payment processors will easily add it to their collection. But the claim that Bitcoin is an outdated pet rock and its value derives *entirely* from a walking-zombie network effect that just needs a little push to collapse is similarly completely wrong.

Crypto-assets like Bitcoin have real cultural and structural advantages that make them powerful assets worth holding and using. Bitcoin is an excellent example of the category, though it's certainly not the only one; other honorable cryptocurrencies do exist, and maximalists have been willing to support and use them. Maximalism is not just Bitcoin-for-the-sake-of-Bitcoin; rather, it's a very genuine realization that most other cryptoassets are scams, and a culture of intolerance is unavoidable and necessary to protect newbies and make sure at least one corner of that space continues to be a corner worth living in.

It's better to mislead ten newbies into avoiding an investment that turns out good than it is to allow a single newbie to get bankrupted by a grifter.

It's better to make your protocol too simple and fail to serve ten low-value short-attention-span gambling applications than it is to make it too complex and fail to serve the central sound money use case that underpins everything else.

And it's better to offend millions by standing aggressively for what you believe in than it is to try to keep everyone happy and end up standing for nothing.

Be brave. Fight for your values. Be a maximalist.

The roads not taken

2022 Mar 29

The Ethereum protocol development community has made a lot of decisions in the early stages of Ethereum that have had a large impact on the project's trajectory. In some cases, Ethereum developers made conscious decisions to improve in some place where we thought that Bitcoin erred. In other places, we were creating something new entirely, and we simply had to come up with *something* to fill in a blank - but there were many somethings to choose from. And in still other places, we had a tradeoff between something more complex and something simpler. Sometimes, we chose the simpler thing, but sometimes, we chose the more complex thing too.

This post will look at some of these forks-in-the-road as I remember them. Many of these features were seriously discussed within core development circles; others were barely considered at all but perhaps really should have been. But even still, it's worth looking at what a different Ethereum might have looked like, and what we can learn from this going forward.

Should we have gone with a much simpler version of proof of stake?

The [Gasper](#) proof of stake that Ethereum is very soon going to merge to is a complex system, but a very powerful system. Some of its properties include:

- **Very strong single-block confirmations** - as soon as a transaction gets included in a block, [usually](#) within a few seconds that block gets solidified to the point that it cannot be reverted unless either a large fraction of nodes are dishonest or there is extreme network latency.
- **Economic finality** - once a block gets *finalized*, it cannot be reverted without the attacker having to lose millions of ETH to being slashed.
- **Very predictable rewards** - validators reliably earn rewards every epoch (6.4 minutes), reducing incentives to pool
- **Support for very high validator count** - unlike most other chains with the above features, the Ethereum beacon chain supports *hundreds of thousands* of validators (eg. Tendermint offers even faster finality than Ethereum, but it [only supports a few hundred](#) validators)

But making a system that has these properties is *hard*. It took [years of research](#), years of [failed experiments](#), and generally took a huge amount of effort. And the final output was pretty complex.
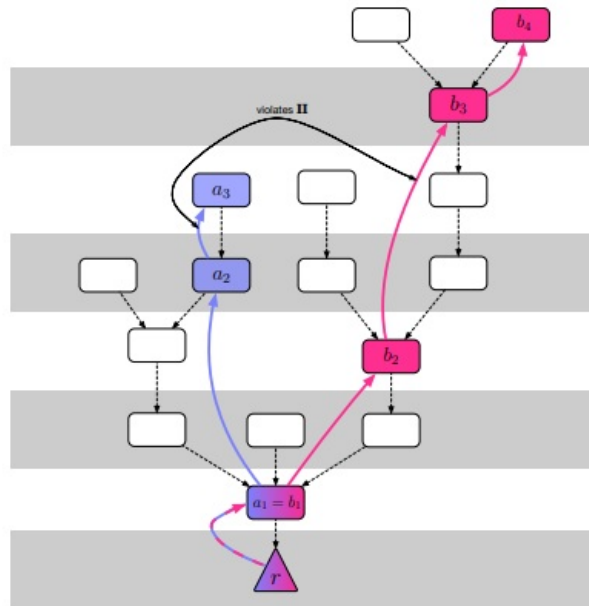
Figure 3: Figure for Theorem 1 (Accountable Safety).

If our researchers did not have to worry so much about consensus and had more brain cycles to spare, then maybe, just maybe, rollups could have been invented in 2016. This brings us to a question: **should we really have had such high standards for our proof of stake, when even a much simpler and weaker version of proof of stake would have been a large improvement over the proof of work status quo?**

Many have the misconception that proof of stake is inherently complex, but in reality there are plenty of proof of stake algorithms that are almost as simple as Nakamoto PoW. NXT proof of stake existed since 2013 and would have been a natural candidate; it had issues but those issues could easily have been patched, and we could have had a reasonably well-working proof of stake from 2017, or even from the beginning. The reason why Gasper is more complex than these algorithms is simply that it *tries to accomplish much more than they do*. But if we had been more modest at the beginning, we could have focused on achieving a more limited set of objectives first.

Proof of stake from the beginning would in my opinion have been a mistake; PoW was helpful in expanding the initial issuance distribution and making Ethereum accessible, as well as encouraging a hobbyist community. But switching to a simpler proof of stake in 2017, or even 2020, could have led to much less environmental damage (and anti-crypto mentality as a result of environmental damage) and a lot more research talent being free to think about scaling. Would we have had to spend a lot of resources on making a better proof of stake eventually? Yes. But it's increasingly looking like we'll end up doing that anyway.

The de-complexification of sharding

Ethereum sharding has been on a very consistent trajectory of becoming less and less complex since the ideas started being worked on in 2014. First, we had complex sharding with built-in execution and cross-shard transactions. Then, we simplified the protocol by moving more responsibilities to the user (eg. in a cross-shard transaction, the user would have to separately pay for gas on both shards). Then, we switched to the rollup-centric roadmap where, from the protocol's point of view, shards are just blobs of data. Finally, with danksharding, the shard fee markets are merged into one, and the final design just looks like a non-sharded chain but where some data availability sampling magic happens behind the scenes to make sharded verification happen.
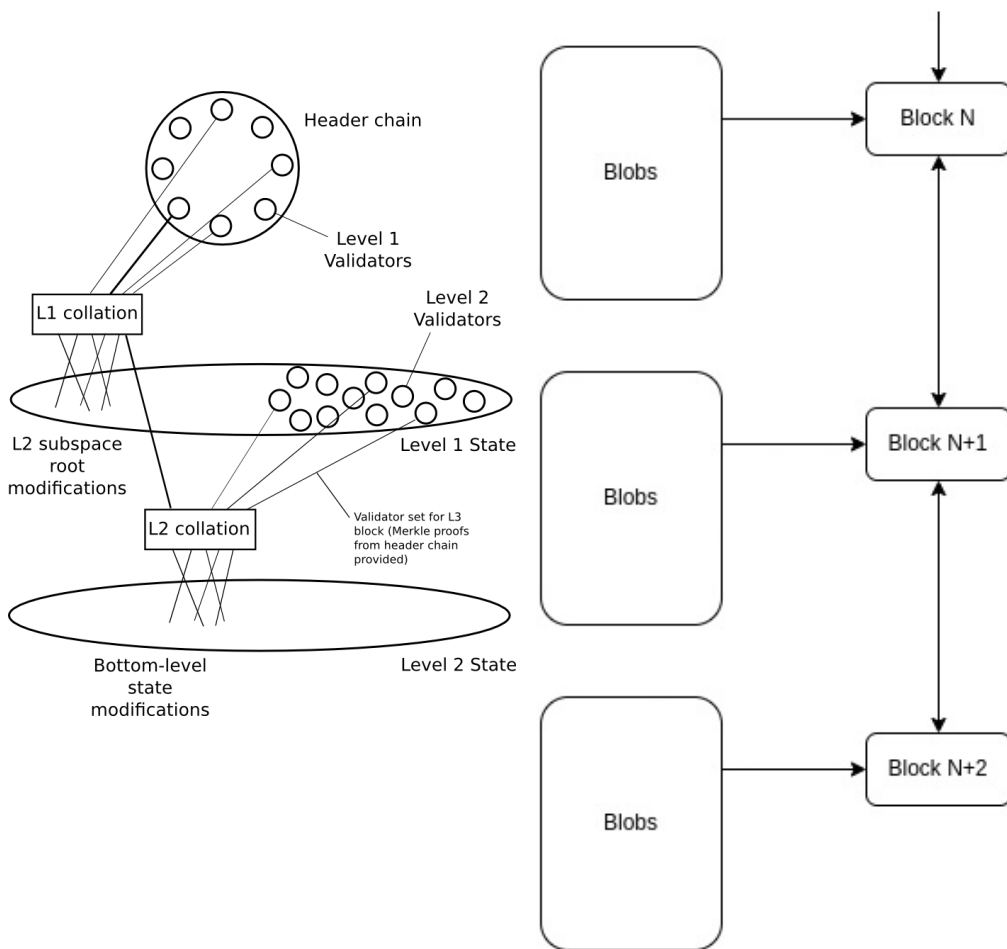
Sharding in 2015   **Sharding in 2022**

But what if we had gone the opposite path? Well, there actually are Ethereum researchers who [heavily explored](#) a much more sophisticated sharding system: shards would be chains, there would be fork choice rules where child chains depend on parent chains, cross-shard messages would get routed by the protocol, validators would be rotated between shards, and even applications would get automatically load-balanced between shards!

The problem with that approach: those forms of sharding are largely just ideas and mathematical models, whereas Danksharding is a complete and almost-ready-for-implementation [spec](#). Hence, given Ethereum's circumstances and constraints, the simplification and de-ambitionization of sharding was, in my opinion, absolutely the right move. That said, the more ambitious research also has a very important role to play: it identifies promising research directions, even the very complex ideas often have "reasonably simple" versions of those ideas that still provide a lot of benefits, and there's a good chance that it will significantly influence Ethereum's protocol development (or even layer-2 protocols) over the years to come.

More or less features in the EVM?

Realistically, the specification of the EVM was basically, with the exception of security auditing, viable for launch by mid-2014. However, over the next few months we continued actively exploring new features that we felt might be really important for a decentralized application blockchain. Some did not go in, others did.

- We **considered [adding a `POST` opcode](#), but decided against it.** The `POST` opcode would have made an *asynchronous* call, that would get executed after the rest of the transaction finishes.
- We **considered [adding an `ALARM` opcode](#), but decided against it.** `ALARM` would have functioned like `POST`, except executing the asynchronous call in some future block, allowing contracts to schedule operations.

- We **added [logs](#)**, which allow contracts to output records that do not touch the state, but could be interpreted by dapp interfaces and wallets. Notably, we also **considered making ETH transfers emit a log, but decided against it** - the rationale being that "people will soon switch to smart contract wallets anyway".
- We **considered expanding [SSTORE to support byte arrays](#), but decided against it**, because of concerns about complexity and safety.
- We **added [precompiles](#)**, contracts which execute specialized cryptographic operations with native implementations at a much cheaper gas cost than can be done in the EVM.
- In the months right after launch, **state rent was [considered](#) [again](#) and [again](#), but was never included**. It was just too complicated. Today, there are much better [state expiry schemes](#) being actively explored, though [stateless verification](#) and [proposer/builder separation](#) mean that it is now a much lower priority.

Looking at this today, most of the decisions to *not* add more features have proven to be very good decisions. There was no obvious reason to add a `POST` opcode. An `ALARM` opcode is actually very difficult to implement safely: what happens if everyone in blocks 1...99999 sets an `ALARM` to execute a lot of code at block 100000? Will that block take hours to process? Will some scheduled operations get pushed back to later blocks? But if that happens, then what guarantees is `ALARM` even preserving? `SSTORE` for byte arrays is difficult to do safely, and would have greatly expanded worst-case witness sizes.

The state rent issue is more challenging: had we actually implemented some kind of state rent from day 1, we would not have had a smart contract ecosystem evolve around a normalized assumption of persistent state. Ethereum would have been harder to build for, but it could have been more scalable and sustainable. At the same time, the state expiry schemes we had back then really were much worse than [what we have now](#). Sometimes, good ideas just take years to arrive at and there is no better way around that.

Alternative paths for `LOG`

`LOG` could have been done differently in two different ways:

1. **We could have made ETH transfers auto-issue a `LOG`**. This would have saved a *lot* of effort and software bug issues for exchanges and many other users, and would have accelerated everyone relying on logs that would have ironically *helped* smart contract wallet adoption.
2. **We could have not bothered with a `LOG` *opcode* at all**, and instead made it an ERC: there would be a standard contract that has a function `submitLog` and uses the [technique from the Ethereum deposit contract](#) to compute a Merkle root of all logs in that block. Either [EIP-2929](#) or block-scoped storage (equivalent to [TSTORE](#) but cleared after the block) would have made this cheap.

We strongly considered (1), but rejected it. The main reason was simplicity: it's easier for logs to *just* come from the `LOG` opcode. We also (very wrongly!) expected most users to quickly migrate to smart contract wallets, which could have logged transfers explicitly using the opcode.

2. was not considered, but in retrospect it was always an option. The main downside of (2) would have been the lack of a Bloom filter mechanism for quickly scanning for logs. But as it turns out, the Bloom filter mechanism is too slow to be user-friendly for dapps anyway, and so these days more and more people [use TheGraph](#) for querying anyway.

On the whole, it seems very possible that *either one* of these approaches would have been superior to the status quo. Keeping `LOG` outside the protocol would have kept things simpler, but if it was inside the protocol auto-logging all ETH transfers would have made it *more useful*.

Today, I would probably favor the eventual abolition of the `LOG` opcode from the EVM.

What if the EVM was something totally different?

There were two natural very different paths that the EVM could have taken:

1. Make the EVM be a **higher-level language**, with built-in constructs for variables, if-statements, loops, etc.
2. Make the EVM be a **copy of some existing VM** (LLVM, WASM, etc)

The first path was never really considered. The attraction of this path is that it could have made compilers simpler, and allowed more developers to code in EVM directly. It could have also made ZK-EVM constructions simpler. The weakness of the path is that it would have made EVM code

structurally more complicated: instead of being a simple list of opcodes in a row, it would have been a more complicated data structure that would have had to be stored somehow. That said, there was a missed opportunity for a best-of-both-worlds: some EVM changes could have given us a lot of those benefits while keeping the basic EVM structure roughly as is: [ban dynamic jumps and add some opcodes designed to support subroutines](#) (see also: [EIP-2315](#)), allow memory access only on 32-byte word boundaries, etc.
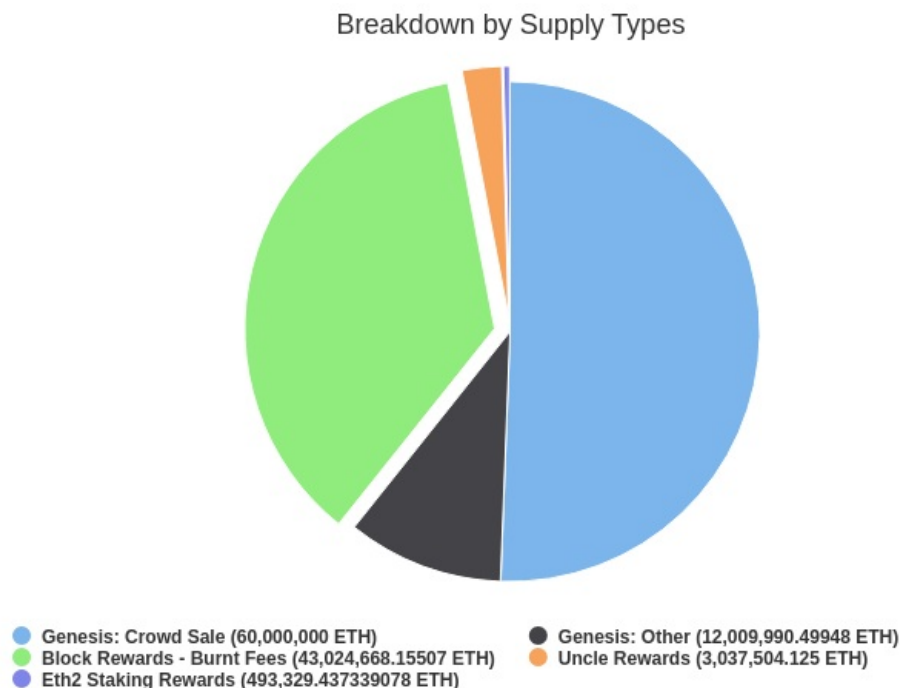
The second path was suggested many times, and rejected many times. The usual argument for it is that it would allow programs to compile from existing languages (C, Rust, etc) into the EVM. The argument against has always been that given Ethereum's unique constraints it would not actually provide any benefits:

- Existing compilers from high-level languages tend to not care about total code size, whereas blockchain code must optimize heavily to cut down every byte of code size
- We need multiple implementations of the VM with a hard requirement that two implementations *never* process the same code differently. Security-auditing and verifying this on code that we did not write would be much harder.
- If the VM specification changes, Ethereum would have to either always update along with it or fall more and more out-of-sync.

Hence, there probably was never a viable path for the EVM that's *radically* different from what we have today, though there are lots of smaller details (jumps, 64 vs 256 bit, etc) that could have led to much better outcomes if they were done differently.

Should the ETH supply have been distributed differently?

The current ETH supply is approximately represented by this [chart from Etherscan](#):

Breakdown by Supply Types

- Genesis: Crowd Sale (60,000,000 ETH)
- Block Rewards - Burnt Fees (43,024,668.15507 ETH)
- Eth2 Staking Rewards (493,329.437339078 ETH)
- Genesis: Other (12,009,990.49948 ETH)
- Uncle Rewards (3,037,504.125 ETH)

About half of the ETH that exists today was sold in an open public [ether sale](#), where anyone could send BTC to a standardized bitcoin address, and the initial ETH supply distribution was computed by an [open-source script](#) that scans the Bitcoin blockchain for transactions going to that address. Most of the remainder was mined. The slice at the bottom, the 12M ETH marked "other", was the "premine" - a piece distributed between the Ethereum Foundation and ~100 early contributors to the Ethereum protocol.

There are two main criticisms of this process:

- **The premine, as well as the fact that the Ethereum Foundation received the sale funds, is not [credibly neutral](#)**. A few recipient addresses were hand-picked through a closed process, and the Ethereum Foundation had to be trusted to not take out loans to recycle funds received furing the sale back into the sale to give itself more ETH (we did not, and no one seriously claims that we have, but even the requirement to be trusted at all offends some).

- **The premine over-rewarded very early contributors, and left too little for later contributors**. 75% of the premine went to rewarding contributors for their work before launch, and post-launch the Ethereum Foundation only had 3 million ETH left. Within 6 months, the need to sell to financially survive decreased that to around 1 million ETH.

In a way, the problems were related: the desire to minimize perceptions of centralization contributed to a smaller premine, and a smaller premine was exhausted more quickly.

This is not the only way that things could have been done. [Zcash](#) has a different approach: a constant 20% of the block reward goes to a set of recipients hard-coded in the protocol, and the set of recipients gets re-negotiated every 4 years (so far this has [happened once](#)). This would have been much more sustainable, but it would have been much more heavily criticized as centralized (the Zcash community seems to be more openly okay with more technocratic leadership than the Ethereum community).

One possible alternative path would be something similar to the "DAO from day 1" route popular among some defi projects today. Here is a possible strawman proposal:

- We agree that for 2 years, a block reward of **2 ETH per block goes into a dev fund**.
- Anyone who purchases ETH in the ether sale could specify a **vote for their preferred distribution** of the dev fund (eg. "1 ETH per block to the Ethereum Foundation, 0.4 ETH to the Consensys research team, 0.2 ETH to Vlad Zamfir...")
- **Recipients that got voted for get a share from the dev fund** equal to the **median of everyone's votes**, scaled so that the total equals 2 ETH per block (median is to prevent self-dealing: if you vote for yourself you get nothing unless you get at least half of other purchasers to mention you)

The sale could be run by a legal entity that promises to distribute the *bitcoin* received during the sale along the same ratios as the ETH dev fund (or burned, if we really wanted to make bitcoiners happy). This probably would have led to the Ethereum Foundation getting a lot of funding, non-EF groups also getting a lot of funding (leading to more ecosystem decentralization), all without breaking credible neutrality one single bit. The main downside is of course that [coin voting really sucks](#), but pragmatically we could have realized that 2014 was still an early and idealistic time and the most serious downsides of coin voting would only start coming into play long after the sale ends.

Would this have been a better idea and set a better precedent? Maybe! Though realistically even if the dev fund had been fully credibly neutral, the people who yell about Ethereum's premine today may well have just started yelling twice as hard about the DAO fork instead.

What can we learn from all this?

In general, **it sometimes feels to me like Ethereum's biggest challenges come from balancing between two visions - a pure and simple blockchain that values safety and simplicity, and a highly performant and functional platform for building advanced applications**. Many of the examples above are just aspects of this: do we have fewer features and be more Bitcoin-like, or more features and be more developer-friendly? Do we worry a lot about making development funding credibly neutral and be more Bitcoin-like, or do we just worry first and foremost about making sure devs are rewarded enough to make Ethereum great?

My personal dream is to try to achieve both visions at the same time - a base layer where the specification becomes *smaller* each year than the year before it, and a powerful developer-friendly advanced application ecosystem centered around layer-2 protocols. That said, getting to such an ideal world takes a long time, and **a more explicit realization that it would take time and we need to think about the roadmap step-by-step would have probably helped us a lot.**

Today, there are a lot of things we cannot change, but there are many things that we still can, and there is still a path solidly open to improving both functionality and simplicity. Sometimes the path is a winding one: we need to add some more complexity first to enable sharding, which in turn enables lots of layer-2 scalability on top. That said, reducing complexity is possible, and Ethereum's history has already demonstrated this:

- [EIP-150](#) made the call stack depth limit no longer relevant, reducing security worries for contract developers.
- [EIP-161](#) made the concept of an "empty account" as something separate from an account whose fields are zero no longer exist.
- [EIP-3529](#) removed part of the refund mechanism and made gas tokens no longer viable.

Ideas in the pipeline, like [Verkle trees](), reduce complexity even further. But the question of how to balance the two visions better in the future is one that we should start more actively thinking about.

How do trusted setups work?

Necessary background: [elliptic curves and elliptic curve pairings](#). See also: [Dankrad Feist's article on KZG polynomial commitments](#).

Special thanks to Justin Drake, Dankrad Feist and Chih-Cheng Liang for feedback and review.

Many cryptographic protocols, especially in the areas of [data availability sampling](#) and [ZK-SNARKs](#) depend on trusted setups. **A trusted setup ceremony is a procedure that is done once to generate a piece of data that must then be used every time some cryptographic protocol is run**. Generating this data requires some secret information; the "trust" comes from the fact that some person or some group of people has to generate these secrets, use them to generate the data, and then publish the data and forget the secrets. But once the data is generated, and the secrets are forgotten, no further participation from the creators of the ceremony is required.
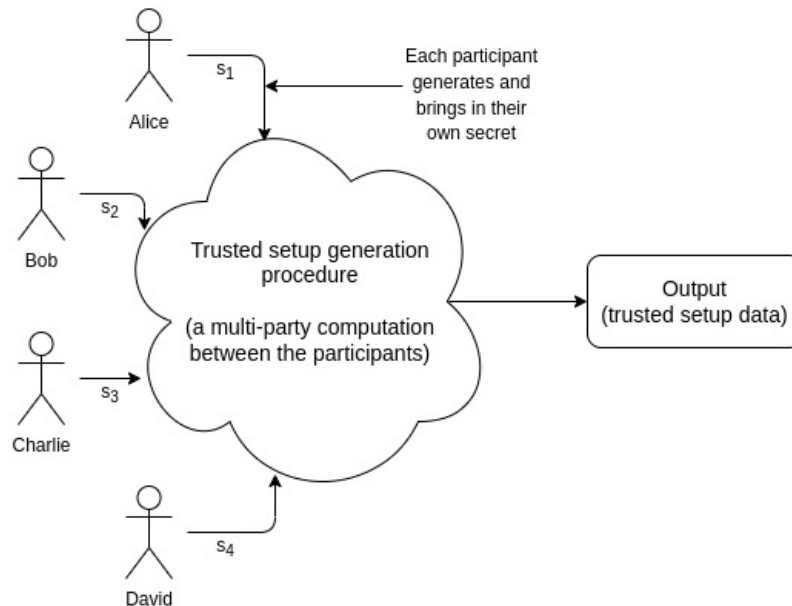
There are many types of trusted setups. The earliest instance of a trusted setup being used in a major protocol is the [original Zcash ceremony](#) in 2016. This ceremony was very complex, and required many rounds of communication, so it could only have six participants. Everyone using Zcash at that point was effectively trusting that at least one of the six participants was honest. More modern protocols usually use the **powers-of-tau** setup, which has a [1-of-N trust model](#) with N typically in the hundreds. That is to say, **hundreds of people participate in generating the data together, and only one of them needs to be honest and not publish their secret for the final output to be secure. Well-executed setups like this are often considered "close enough to trustless" in practice**.

This article will explain how the KZG setup works, why it works, and the future of trusted setup protocols. Anyone proficient in code should also feel free to follow along this code implementation: https://github.com/ethereum/research/blob/master/trusted_setup/trusted_setup.py.

What does a powers-of-tau setup look like?

A powers-of-tau setup is made up of two series of elliptic curve points that look as follows:

$$[G_1, G_1 * s, G_1 * s^2 ... G_1 * s^{n_1-1}]$$

$$[G_2, G_2 * s, G_2 * s^2 ... G_2 * s^{n_2-1}]$$

G_1 and G_2 are the standardized generator points of the two elliptic curve groups; in BLS12-381, G_1 points are (in compressed form) 48 bytes long and G_2 points are 96 bytes long. n_1 and n_2 are the lengths of the G_1 and G_2 sides of the setup. Some protocols require $n_2 = 2$, others require n_1 and n_2 to both be large, and some are in the middle (eg. Ethereum's data availability sampling in its current form requires $n_1 = 4096$ and $n_2 = 16$). s is the secret that is used to generate the points, and needs to be forgotten.

To make a KZG commitment to a polynomial $P(x) = \sum_i c_i x^i$, we simply take a linear combination $\sum_i c_i S_i$, where $S_i = G_1 * s^i$ (the elliptic curve points in the trusted setup). The G_2 points in the setup are used to verify evaluations of polynomials that we make commitments to; I won't go into verification here in more detail, though [Dankrad does in his post](#).

Intuitively, what value is the trusted setup providing?

It's worth understanding what is philosophically going on here, and why the trusted setup is providing value. A polynomial commitment is committing to a piece of size-N data with a size $O(1)$ object (a single elliptic curve point). We *could* do this with a plain Pedersen commitment: just set the S_i values to be N random elliptic curve points that have no known relationship with each other, and commit to polynomials with $\sum_i c_i S_i$ as before. And in fact, this is exactly what [IPA evaluation proofs](#) do.

However, any IPA-based proofs take $O(N)$ time to verify, and there's an unavoidable reason why: a commitment to a polynomial $P(x)$ using the base points $[S_0, S_1 ... S_i ... S_{n-1}]$ would commit to a different polynomial if we use the base points $[S_0, S_1 ... (S_i * 2) ... S_{n-1}]$.
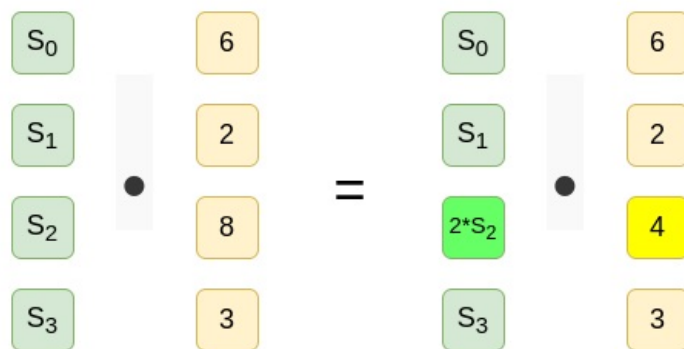
A valid commitment to the polynomial $3x^3 + 8x^2 + 2x + 6$ under one set of base points is a valid commitment to $3x^3 + 4x^2 + 2x + 6$ under a different set of base points.

If we want to make an IPA-based *proof* for some statement (say, that this polynomial evaluated at $x = 10$ equals 3826), the proof should pass with the first set of base points and fail with the second. Hence, whatever the proof verification procedure is cannot avoid somehow taking into account each and every one of the S_i values, and so it unavoidably takes $O(N)$ time.

**But with a trusted setup, there is a hidden mathematical relationship between the points**. It's guaranteed that $S_{i+1} = s * S_i$ with the same factor s between any two adjacent points. If $[S_0, S_1 ... S_i ... S_{n-1}]$ is a valid setup, the "edited setup" $[S_0, S_1 ... (S_i * 2) ... S_{n-1}]$ *cannot also be a valid setup*. **Hence, we don't need $O(n)$ computation; instead, we take advantage of this mathematical relationship to verify anything we need to verify in constant time.**

However, the mathematical relationship has to remain secret: if s is known, then anyone could come up with a commitment that stands for many different polynomials: if C commits to $P(x)$, it also commits to $\frac{P(x) * x}{s}$, or $P(x) - x + s$, or many other things. This would completely break all applications of polynomial commitments. **Hence, while some secret s must have existed at one point to make possible the mathematical link between the S_i values that enables efficient verification, the s must also have been forgotten.**

How do multi-participant setups work?

It's easy to see how one participant can generate a setup: just pick a random s, and generate the elliptic curve points using that s. But a single-participant trusted setup is insecure: you have to trust one specific person!

The solution to this is multi-participant trusted setups, where by "multi" we mean *a lot* of participants: over 100 is normal, and for smaller setups it's possible to get over 1000. Here is how a multi-participant powers-of-tau setup works.
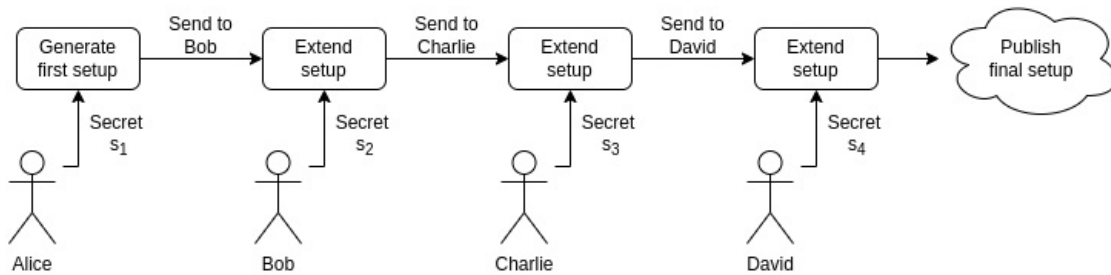
Take an existing setup (note that you don't know s, you just know the points):

$([G\_1, G\_1 * s, G\_1 * s^2 ... G\_1 * s^{n\_1-1}])$

$([G\_2, G\_2 * s, G\_2 * s^2 ... G\_2 * s^{n\_2-1}])$

Now, choose your own random secret t. Compute:

$([G\_1, (G\_1 * s) * t, (G\_1 * s^2) * t^2 ... (G\_1 * s^{n\_1-1}) * t^{n\_2-1}])$

$([G\_2, (G\_2 * s) * t, (G\_2 * s^2) * t^2 ... (G\_2 * s^{n\_2-1}) * t^{n\_2-1}])$

Notice that this is equivalent to:

$([G\_1, G\_1 * (st), G\_1 * (st)^2 ... G\_1 * (st)^{n\_1-1}])$

$([G\_2, G\_2 * (st), G\_2 * (st)^2 ... G\_2 * (st)^{n\_2-1}])$

That is to say, you've created a valid setup with the secret $s * t$! You never give your t to the previous participants, and the previous participants never give you their secrets that went into s. And as long as any one of the participants is honest and does not reveal their part of the secret, the combined secret does not get revealed. In particular, finite fields have the property that if you know know s but not t, and t is securely randomly generated, then you know *nothing* about $s*t$!

Verifying the setup

To verify that each participant actually participated, each participant can provide a proof that consists of (i) the $G\_1 * s$ point that they received and (ii) $G\_2 * t$, where t is the secret that they introduce. The list of these proofs can be used to verify that the final setup combines together all the secrets (as opposed to, say, the last participant just forgetting the previous values and outputting a setup with just their own secret, which they keep so they can cheat in any protocols that use the setup).
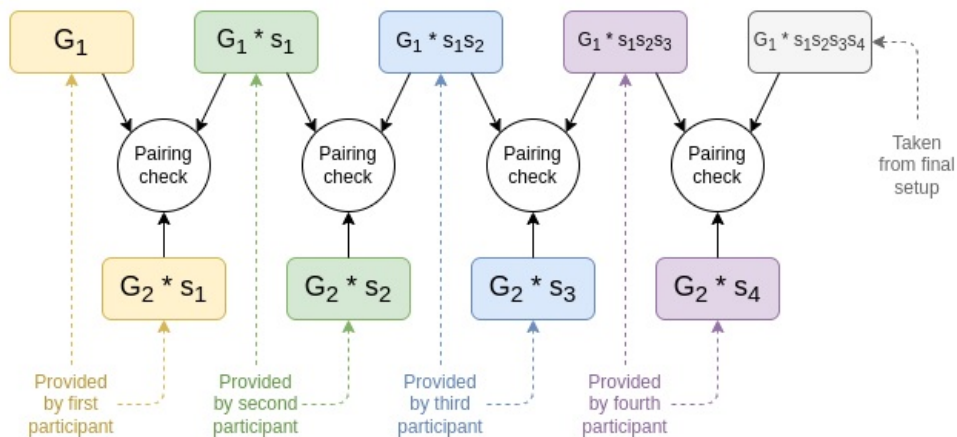
s_1 is the first participant's secret, s_2 is the second participant's secret, etc. The pairing check at each step proves that the setup at each step actually came from a combination of the setup at the previous step and a new secret known by the participant at that step.

Each participant should reveal their proof on some publicly verifiable medium (eg. personal website, transaction from their .eth address, Twitter). Note that this mechanism does *not* prevent someone from claiming to have participated at some index where someone else has (assuming that other person has revealed their proof), but it's generally considered that this does not matter: if someone is willing to lie about having participated, they would also be willing to lie about having deleted their secret. As long as at least one of the people who publicly claim to have participated is honest, the setup is secure.

In addition to the above check, we also want to verify that all the powers in the setup are correctly constructed (ie. they're powers of the same secret). To do this, we *could* do a series of pairing checks, verifying that $e(S_{i+1}, G_2) = e(S_i, T_1)$ (where T_1 is the $G_2 * s$ value in the setup) for every i. This verifies that the factor between each S_i and S_{i+1} is the same as the factor between T_1 and G_2. We can then do the same on the G_2 side.

But that's a lot of pairings and is expensive. Instead, we take a random linear combination $L_1 = \sum_{i=0}^{n_1-2} r_i S_i$, and the same linear combination shifted by one: $L_2 = \sum_{i=0}^{n_1-2} r_i S_{i+1}$. We use a single pairing check to verify that they match up: $e(L_2, G_2) = e(L_1, T_1)$.

We can even combine the process for the G_1 side and the G_2 side together: in addition to computing L_1 and L_2 as above, we also compute $L_3 = \sum_{i=0}^{n_2-2} q_i T_i$ (q_i is another set of random coefficients) and $L_4 = \sum_{i=0}^{n_2-2} q_i T_{i+1}$, and check $e(L_2, L_3) = e(L_1, L_4)$.

Setups in Lagrange form

In many use cases, you don't want to work with polynomials *in coefficient form* (eg. $P(x) = 3x^3 + 8x^2 + 2x + 6$), you want to work with polynomials in *evaluation form* (eg. $P(x)$ is the polynomial that evaluates to $[19, 146, 9, 187]$ on the domain $[1, 189, 336, 148]$ modulo 337). Evaluation form has many advantages (eg. you can multiply and sometimes divide polynomials in $O(N)$ time) and you can even use it to [evaluate in $O(N)$ time](#). In particular, [data availability sampling](#) expects the blobs to be in evaluation form.

To work with these cases, it's often convenient to convert the trusted setup to evaluation form. This would allow you to take the evaluations ($[19, 146, 9, 187]$ in the above example) and use them to compute the commitment directly.

This is done most easily with a [Fast Fourier transform (FFT)](#), but passing the curve points as input instead of numbers. I'll avoid repeating a full detailed explanation of FFTs here, but [here is an implementation](#); it is actually not that difficult.

The future of trusted setups

Powers-of-tau is not the only kind of trusted setup out there. Some other notable (actual or potential) trusted setups include:

- The more complicated setups in older ZK-SNARK protocols (eg. see [here](#)), which are sometimes still used (particularly [Groth16](#)) because verification is cheaper than PLONK.
- Some cryptographic protocols (eg. **[DARK](#)**) depend on **hidden-order groups**, groups where it is not known what number an element can be multiplied by to get the zero element. Fully trustless versions of this exist (see: [class groups](#)), but by far the most efficient version uses RSA groups (powers of x mod $n = pq$ where p and q are not known). Trusted setup ceremonies for this with 1-of-n trust assumptions [are possible](#), but are very complicated to implement.
- If/when **[indistinguishability obfuscation](#)** becomes viable, many protocols that depend on it will involve someone creating and publishing an obfuscated program that does something with a hidden internal secret. This is a trusted setup: the creator(s) would need to possess the secret to create the program, and would need to delete it afterwards.

Cryptography continues to be a rapidly evolving field, and how important trusted setups are could easily change. It's possible that techniques for working with IPAs and Halo-style ideas will improve to the point where KZG becomes outdated and unnecessary, or that quantum computers will make anything based on elliptic curves non-viable ten years from now and we'll be stuck working with trusted-setup-free hash-based protocols. It's also possible that what we can do with KZG will improve even faster, or that a new area of cryptography will emerge that depends on a different kind of trusted setup.

To the extent that trusted setup ceremonies are necessary, it is important to remember that **not all trusted setups are created equal**. [176 participants](#) is better than 6, and 2000 would be even better. A ceremony small enough that it can be run inside a browser or phone application (eg. the [ZKopru setup is web-based](#)) could attract far more participants than one that requires running a complicated software package. Every ceremony should ideally have participants running multiple independently built software implementations and running different operating systems and environments, to reduce [common mode failure](#) risks. Ceremonies that require only one round of interaction per participant (like powers-of-tau) are far better than multi-round ceremonies, both due to the ability to support far more participants and due to the greater ease of writing multiple implementations. Ceremonies should ideally be *universal* (the output of one ceremony being able to support a wide range of protocols). These are all things that we can and should keep working on, to ensure that trusted setups can be as secure and as trusted as possible.

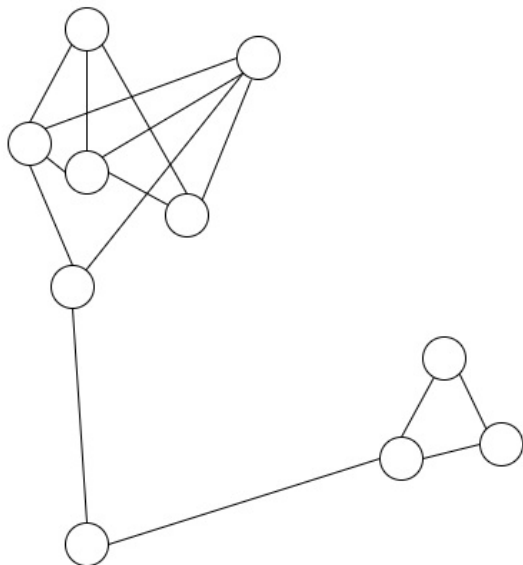# Encapsulated vs systemic complexity in protocol design

2022 Feb 28

One of the main [goals of Ethereum protocol design](#) is to minimize complexity: make the protocol as simple as possible, while still making a blockchain that can [do what an effective blockchain needs to do](#). The Ethereum protocol is far from perfect at this, especially since much of it was designed in 2014-16 when we understood much less, but we nevertheless make an active effort to reduce complexity whenever possible.

One of the challenges of this goal, however, is that complexity is difficult to define, and sometimes, you have to trade off between two choices that introduce different kinds of complexity and have different costs. How do we compare?

One powerful intellectual tool that allows for more nuanced thinking about complexity is to draw a distinction between what we will call **encapsulated complexity** and **systemic complexity**.
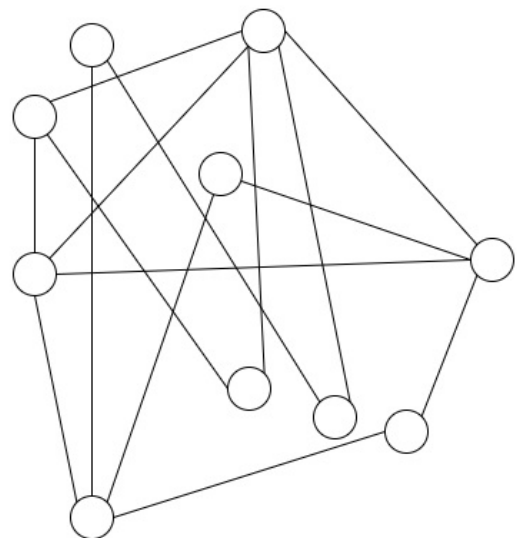
Encapsulated complexity occurs when there is a system with sub-systems that are internally complex, but that present a simple "interface" to the outside. Systemic complexity occurs when the different parts of a system can't even be cleanly separated, and have complex interactions with each other.

Here are a few examples.

BLS signatures vs Schnorr signatures

[BLS signatures](#) and [Schnorr signatures](#) are two popular types of cryptographic signature schemes that can be made with elliptic curves.

BLS signatures appear mathematically very simple:

Signing: $\sigma = H(m) * k$

Verifying: $e([1], \sigma) \stackrel{?}{=} e(H(m), K)$

H is a hash function, m is the message, and k and K are the private and public keys. So far, so simple. However, the true complexity is hidden inside the definition of the e function: [elliptic curve pairings](#), one of the most devilishly hard-to-understand pieces of math in all of cryptography.

Now, consider Schnorr signatures. Schnorr signatures rely only on basic [elliptic curves](#). But the signing and verification logic is somewhat more complex:

Signing [edit]

To sign a message, M:

- Choose a random k from the allowed set.
- Let $r = g^k$.
- Let $e = H(r \parallel M)$, where $\parallel$ denotes concatenation and r is represented as a bit string.
- Let $s = k - xe$.

The signature is the pair, (s, e).

Note that $s, e \in \mathbb{Z}_q$; if $q < 2^{160}$, then the signature representation can fit into 40 bytes.

Verifying [edit]

- Let $r_v = g^s y^e$
- Let $e_v = H(r_v \parallel M)$

If $e_v = e$ then the signature is verified.

So... which type of signature is "simpler"? It depends what you care about! BLS signatures have a huge amount of technical complexity, but the complexity is all buried within the definition of the e function. If you treat the e function as a black box, BLS signatures are actually really easy. Schnorr signatures, on the other hand, have less *total* complexity, but they have more pieces that could interact with the outside world in tricky ways.

For example:

- Doing a BLS multi-signature (a combined signature from two keys k_1 and k_2) is easy: just take $\sigma_1 + \sigma_2$. But a Schnorr multi-signature requires two rounds of interaction, and there are tricky [key cancellation attacks](#) that need to be dealt with.
- Schnorr signatures require random number generation, BLS signatures do not.

Elliptic curve pairings in general are a powerful "complexity sponge" in that they contain large amounts of encapsulated complexity, but enable solutions with much less systemic complexity. This is also true in the area of polynomial commitments: compare the [simplicity of KZG commitments](#) (which require pairings) to the much more complicated internal logic of [inner product arguments](#) (which do not).

Cryptography vs cryptoeconomics

One important design choice that appears in many blockchain designs is that of cryptography versus cryptoeconomics. Often (eg. in [rollups](#)) this comes in the form of a choice between **validity proofs** (aka. ZK-SNARKs) and **fraud proofs**.

ZK-SNARKs are complex technology. While [the basic ideas](#) behind how they work can be explained in a single post, actually implementing a ZK-SNARK to verify some computation involves many times more complexity than the computation itself (hence why ZK-SNARKs for the EVM are [still under development](#) while fraud proofs for the EVM are [already in the testing stage](#)). Implementing a ZK-SNARK effectively involves circuit design with special-purpose optimization, working with unfamiliar programming languages, and many other challenges. Fraud proofs, on the other hand, are inherently simple: if someone makes a challenge, you just directly run the computation on-chain. For efficiency, a binary-search scheme is sometimes added, but even that doesn't add too much complexity.

But while ZK-SNARKs are complex, their complexity is *encapsulated complexity*. The relatively light complexity of fraud proofs, on the other hand, is *systemic*. Here are some examples of systemic complexity that fraud proofs introduce:

- They require careful incentive engineering to avoid the [verifier's dilemma](#).
- If done in-consensus, they require extra transaction types for the fraud proofs, along with reasoning about what happens if many actors compete to submit a fraud proof at the same time.
- They depend on a synchronous network.
- They allow censorship attacks to be also used to commit theft.
- Rollups based on fraud proofs require liquidity providers to support instant withdrawals.

For these reasons, even from a complexity perspective purely cryptographic solutions based on ZK-SNARKs are likely to be long-run safer: ZK-SNARKs have are more complicated parts that *some* people have to think about, but they have fewer dangling caveats that *everyone* has to think about.

Miscellaneous examples

- **Proof of work (Nakamoto consensus)** - low encapsulated complexity, as the mechanism is extremely simple and easy to understand, but higher systemic complexity (eg. selfish mining attacks).
- **Hash functions** - high encapsulated complexity, but very easy-to-understand properties so low systemic complexity.
- **Random shuffling algorithms** - shuffling algorithms can either be internally complicated (as in Whisk) but lead to easy-to-understand guarantees of strong randomness, or internally simpler but lead to randomness properties that are weaker and more difficult to analyze (systemic complexity).
- **Miner extractable value (MEV)** - a protocol that is powerful enough to support complex transactions can be fairly simple internally, but those complex transactions can have complex systemic effects on the protocol's incentives by contributing to the incentive to propose blocks in very irregular ways.
- **Verkle trees** - Verkle trees do have some encapsulated complexity, in fact quite a bit more than plain Merkle hash trees. Systemically, however, Verkle trees present the exact same relatively clean-and-simple interface of a key-value map. The main systemic complexity "leak" is the possibility of an attacker manipulating the tree to make a particular value have a very long branch; but this risk is the same for both Verkle trees and Merkle trees.

How do we make the tradeoff?

Often, the choice with less encapsulated complexity is also the choice with less systemic complexity, and so there is one choice that is obviously simpler. But at other times, you have to make a hard choice between one type of complexity and the other. What should be clear at this point is that **complexity is less dangerous if it is encapsulated**. The risks from complexity of a system are not a simple function of how long the specification is; a small 10-line piece of the specification that interacts with every other piece adds more complexity than a 100-line function that is otherwise treated as a black box.

However, there are limits to this approach of preferring encapsulated complexity. Software bugs can occur in any piece of code, and as it gets bigger the probability of a bug approaches 1. Sometimes, when you need to interact with a sub-system in an unexpected and new way, **complexity that was originally encapsulated can become systemic**.

One example of the latter is Ethereum's current two-level state tree, which features a tree of account objects, where each account object in turn has its own storage tree.
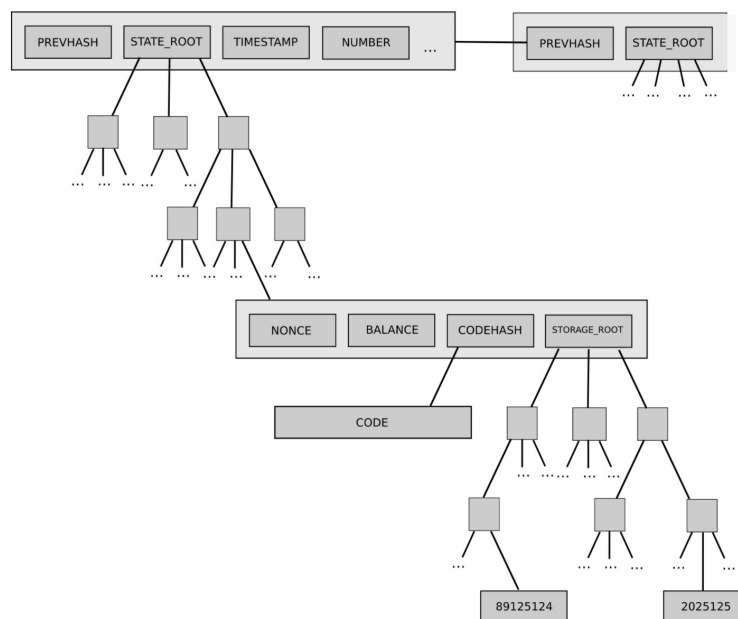
This tree structure is complex, but at the beginning the complexity seemed to be well-encapsulated: the rest of the protocol interacts with the tree as a key/value store that you can read and write to, so we don't have to worry about how the tree is structured.

Later, however, the complexity turned out to have systemic effects: the ability of accounts to have arbitrarily large storage trees meant that there was no way to reliably expect a particular slice of the state (eg. "all accounts starting with 0x1234") to have a predictable size. This makes it harder to split up the state into pieces, complicating the design of syncing protocols and attempts to distribute the storage process. **Why did encapsulated complexity become systemic? Because the interface changed.** The fix? The current proposal to move to Verkle trees also includes a move to a well-balanced single-layer design for the tree,

Ultimately, which type of complexity to favor in any given situation is a question with no easy answers. The

best that we can do is to have an attitude of moderately favoring encapsulated complexity, but not too much, and exercise our judgement in each specific case. Sometimes, a sacrifice of a little bit of systemic complexity to allow a great reduction of encapsulated complexity really is the best thing to do. And other times, you can even misjudge what is encapsulated and what isn't. Each situation is different.

Soulbound

2022 Jan 26

One feature of World of Warcraft that is second nature to its players, but goes mostly undiscussed outside of gaming circles, is the concept of *soulbound* items. A soulbound item, once picked up, cannot be transferred or sold to another player.

Most very powerful items in the game are soulbound, and typically require completing a complicated quest or killing a very powerful monster, usually with the help of anywhere from four to thirty nine other players. Hence, in order to get your character anywhere close to having the best weapons and armor, you have no choice but to participate in killing some of these extremely difficult monsters yourself.

The purpose of the mechanic is fairly clear: it keeps the game challenging and interesting, by making sure that to get the best items you have to actually go and do the hard thing and figure out how to kill the dragon. You can't just go kill boars ten hours a day for a year, get thousands of gold, and buy the epic magic armor from other players who killed the dragon for you.

Of course, the system is very imperfect: you could just pay a team of professionals to kill the dragon with you and let you collect the loot, or even outright buy a character on a secondary market, and do this all with out-of-game US dollars so you don't even have to kill boars. But even still, it makes for a much better game than every item always having a price.

What if NFTs could be soulbound?

NFTs in their current form have many of the same properties as rare and epic items in a massively multiplayer online game. They have social signaling value: people who have them can show them off, and there's more and more tools precisely to help users do that. Very recently, Twitter started rolling out an integration that allows users to show off their NFTs on their picture profile.

But what exactly are these NFTs signaling? Certainly, one part of the answer is some kind of skill in acquiring NFTs and knowing which NFTs to acquire. But because NFTs are tradeable items, another big part of the answer inevitably becomes that NFTs are about signaling wealth.

CryptoPunks are now regularly being sold for many millions of dollars, and they are not even [the most expensive NFTs](#) out there. Image source [here](#).

If someone shows you that they have an NFT that is obtainable by doing X, you can't tell whether they did X themselves or whether they just paid someone else to do X. Some of the time this is not a problem: for an NFT supporting a charity, someone buying it off the secondary market is sacrificing their own funds for the cause and they are helping the charity by contributing to others' incentive to buy the NFT, and so there is no reason to discriminate against them. And indeed, a lot of good can come from charity NFTs alone. But what if we want to create NFTs that are not just about who has the most money, and that actually try to signal something else?

Perhaps the best example of a project trying to do this is [POAP](#), the "proof of attendance protocol". POAP is a standard by which projects can send NFTs that represent the idea that the recipient personally participated in some event.

Part of [my own POAP collection](#), much of which came from the events that I attended over the years.

POAP is an excellent example of an NFT that works better if it could be soulbound. If someone is looking at your POAP, they are not interested in whether or not you paid someone who attended some event. They are interested in whether or not *you personally* attended that event. Proposals to

put certificates (eg. driver's licenses, university degrees, proof of age) on-chain face a similar problem: they would be much less valuable if someone who doesn't meet the condition themselves could just go buy one from someone who does.

While transferable NFTs have their place and can be really valuable on their own for supporting artists and charities, there is also a large and underexplored design space of what *non-transferable* NFTs could become.

What if governance rights could be soulbound?

This is a topic I have written about ad nauseam (see [1] [2] [3] [4] [5]), but it continues to be worth repeating: **there are very bad things that can easily happen to governance mechanisms if governance power is easily transferable**. This is true for two primary types of reasons:

- If the goal is for governance power *to be widely distributed*, then transferability is counterproductive as concentrated interests are more likely to buy the governance rights up from everyone else.
- If the goal is for governance power *to go to the competent*, then transferability is counterproductive because nothing stops the governance rights from being bought up by the determined but incompetent.

If you take the proverb that "those who most want to rule people are those least suited to do it" seriously, then you should be suspicious of transferability, precisely because transferability makes governance power flow away from the meek who are most likely to provide valuable input to governance and toward the power-hungry who are most likely to cause problems.

So what if we try to make governance rights non-transferable? What if we try to make a CityDAO where more voting power goes to the people who actually live in the city, or at least is reliably democratic and avoids undue influence by whales hoarding a large number of citizen NFTs? What if DAO governance of blockchain protocols could somehow make governance power conditional on participation? Once again, a large and fruitful design space opens up that today is difficult to access.

Implementing non-transferability in practice

POAP has made the technical decision to not block transferability of the POAPs themselves. There are good reasons for this: users might have a good reason to want to migrate all their assets from one wallet to another (eg. for security), and the security of non-transferability implemented "naively" is not very strong anyway because users could just create a wrapper account that holds the NFT and then sell the ownership of that.

And indeed, there have been quite a few cases where POAPs have frequently been bought and sold when an economic rationale was there to do so. Adidas recently released a POAP for free to their fans that could give users priority access at a merchandise sale. What happened? Well, of course, many of the POAPs were quickly transferred to the highest bidder.

adidas Originals: our future started here

📅 17-Nov-2021   🖥 Virtual event

For those of you in search of new playgrounds where possibilities are limitless, we invite you to join us as we voyage into the Metaverse.
This digital collectible is our way of rewarding you

https://confirmed.onelink.me/mzYA/589f9ddf

🗒 SUPPLY
3475

🔥 POWER
11030

🔁 TRANSFERS
4204

More transfers than items. And [not the only time](#).

To solve this problem, the POAP team is suggesting that developers who care about non-transferability implement checks on their own: they could check on-chain if the current owner is the same address as the original owner, and they could add more sophisticated checks over time if deemed necessary. This is, for now, a more future-proof approach.

Perhaps the one NFT that is the most robustly non-transferable today is the [proof-of-humanity attestation](#). Theoretically, anyone can create a proof-of-humanity profile with a smart contract account that has transferable ownership, and then sell that account. But the proof-of-humanity protocol has a [revocation feature](#) that allows the original owner to make a video asking for a profile to be removed, and a [Kleros](#) court decides whether or not the video was from the same person as the original creator. Once the profile is successfully removed, they can re-apply to make a new profile. Hence, if you buy someone else's proof-of-humanity profile, your possession can be very quickly taken away from you, making transfers of ownership non-viable. Proof-of-humanity profiles are de-facto soulbound, and infrastructure built on top of them could allow for on-chain items in general to be soulbound to particular humans.

Can we limit transferability without going all the way and basing everything on proof of humanity? It becomes harder, but there are medium-strength approaches that are probably good enough for some use cases. Making an NFT bound to an ENS name is one simple option, if we assume that users care enough about their ENS names that they are not willing to transfer them. For now, what we're likely to see is a spectrum of approaches to limit transferability, with different projects choosing different tradeoffs between security and convenience.

Non-transferability and privacy

Cryptographically strong privacy for transferable assets is fairly easy to understand: you take your coins, put them into [tornado.cash](#) or a similar platform, and withdraw them into a fresh account. But how can we add privacy for soulbound items where you cannot just move them into a fresh account or even a smart contract? If proof of humanity starts getting more adoption, privacy becomes even more important, as the alternative is all of our activity being mapped on-chain directly to a human face.

Fortunately, a few fairly simple technical options are possible:

- Store the item at an address which is the hash of (i) an index, (ii) the recipient address and (iii) a secret belonging to the recipient. You could reveal your secret to an interface that would then scan for all possible items that belong to your, but no one without your secret could see which items are yours.
- Publish a hash of a bunch of items, and give each recipient their Merkle branch.

- If a *smart contract* needs to check if you have an item of some type, you can provide a ZK-SNARK.

Transfers could be done on-chain; the simplest technique may just be a transaction that calls a factory contract to make the old item invalid and the new item valid, using a ZK-SNARK to prove that the operation is valid.

Privacy is an important part of making this kind of ecosystem work well. In some cases, the underlying thing that the item is representing is already public, and so there is no point in trying to add privacy. But in many other cases, users would not want to reveal everything that they have. If, one day in the future, being vaccinated becomes a POAP, one of the worst things we could do would be to create a system where the POAP is automatically advertised for everyone to see and everyone has no choice but to let their medical decision be influenced by what would look cool in their particular social circle. Privacy being a core part of the design can avoid these bad outcomes and increase the chance that we create something great.

From here to there

A common criticism of the "web3" space as it exists today is how money-oriented everything is. People celebrate the ownership, and outright waste, of large amounts of wealth, and this limits the appeal and the long-term sustainability of the culture that emerges around these items. There are of course important benefits that even financialized NFTs can provide, such as funding artists and charities that would otherwise go unrecognized. However, there are limits to that approach, and a lot of underexplored opportunity in trying to go beyond financialization. Making more items in the crypto space "soulbound" can be one path toward an alternative, where NFTs can represent much more of who you are and not just what you can afford.

However, there are technical challenges to doing this, and an uneasy "interface" between the desire to limit or prevent transfers and a blockchain ecosystem where so far all of the standards are designed around maximum transferability. Attaching items to "identity objects" that users are either unable (as with proof-of-humanity profiles) or unwilling (as with ENS names) to trade away seems like the most promising path, but challenges remain in making this easy-to-use, private and secure. We need more effort on thinking through and solving these challenges. If we can, this opens a much wider door to blockchains being at the center of ecosystems that are collaborative and fun, and not just about money.

The bulldozer vs vetocracy political axis

2021 Dec 19

Typically, attempts to collapse down political preferences into a few dimensions focus on two primary dimensions: "authoritarian vs libertarian" and "left vs right". You've probably seen political compasses like this:
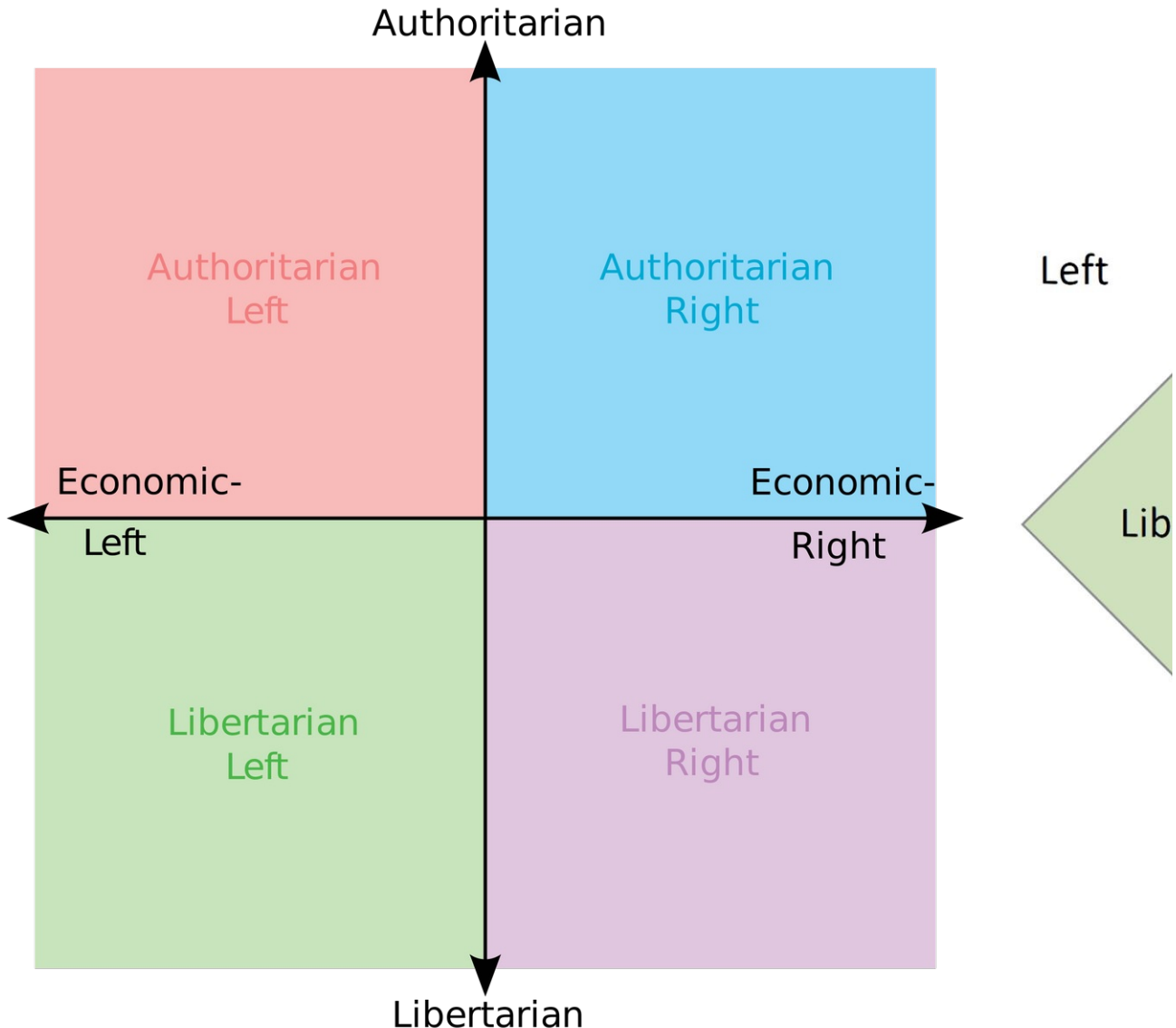
There have been many variations on this, and even an entire subreddit dedicated to memes based on these charts. I even made a spin on the concept myself, with this "meta-political compass" where at each point on the compass there is a smaller compass depicting what the people at that point on the compass see the axes of the compass as being.

Of course, "authoritarian vs libertarian" and "left vs right" are both incredibly un-nuanced gross oversimplifications. But us puny-brained human beings do not have the capacity to run anything close to accurate simulations of humanity inside our heads, and so sometimes incredibly un-nuanced gross oversimplifications are something we need to understand the world. But what if there are other incredibly un-nuanced gross oversimplifications worth exploring?

Enter the bulldozer vs vetocracy divide

Let us consider a political axis defined by these two opposing poles:

- **Bulldozer**: single actors can do important and meaningful, but potentially risky and disruptive, things without asking for permission
- **Vetocracy**: doing anything potentially disruptive and controversial requires getting a sign-off from a large number of different and diverse actors, any of whom could stop it

Note that this is not the same as either authoritarian vs libertarian *or* left vs right. You can have vetocratic authoritarianism, the bulldozer left, or any other combination. Here are a few examples:
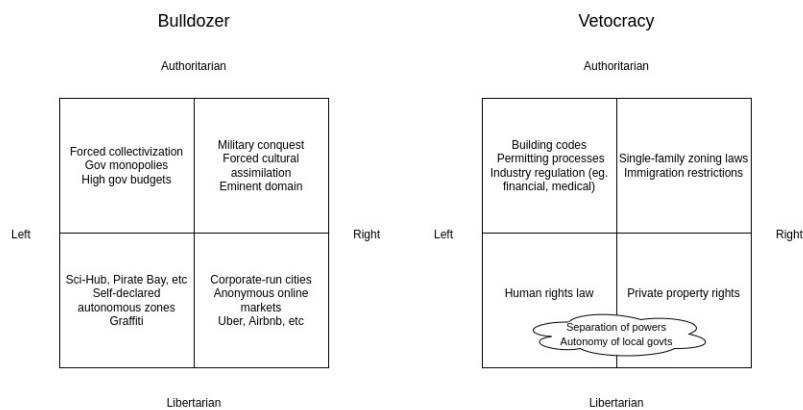
The key difference between authoritarian bulldozer and authoritarian vetocracy is this: is the government more likely to fail by *doing bad things* or by *preventing good things from happening*? Similarly for libertarian bulldozer vs vetocracy: are private actors more likely to fail by doing bad things, or by standing in the way of needed good things?

Sometimes, I hear people complaining that eg. the United States (but other countries too) is falling behind because too many people use freedom as an excuse to prevent needed reforms from happening. But is the problem really freedom? Isn't, say, restrictive housing policy preventing GDP from rising by 36% an example of the problem precisely being *people not having enough freedom* to build structures on their own land? Shifting the argument over to saying that there is too much *vetocracy*, on the other hand, makes the argument look much less confusing: individuals excessively blocking governments and governments excessively blocking individuals are not opposites, but rather two sides of the same coin.

And indeed, recently there has been a bunch of political writing pointing the finger straight at vetocracy as a source of many huge problems:

- https://astralcodexten.substack.com/p/ezra-klein-on-vetocracy
- https://www.vox.com/2020/4/22/21228469/marc-andreessen-build-government-coronavirus
- https://www.vox.com/2016/10/26/13352946/francis-fukuyama-ezra-klein
- https://www.politico.com/news/magazine/2019/11/29/penn-station-robert-caro-073564

And on the other side of the coin, people are often confused when politicians who normally do not respect human rights suddenly appear very pro-freedom in their love of Bitcoin. Are they libertarian, or are they authoritarian? In this framework, the answer is simple: they're bulldozers, with all the benefits and risks that that side of the spectrum brings.

What is vetocracy good for?

Though the change that cryptocurrency proponents seek to bring to the world is often bulldozery, cryptocurrency governance internally is often quite vetocratic. Bitcoin governance famously makes it very difficult to make changes, and some core "constitutional norms" (eg. the 21 million coin limit) are considered so inviolate that many Bitcoin users consider a chain that violates that rule to be by-definition not Bitcoin, regardless of how much support it has.

Ethereum protocol *research* is sometimes bulldozery in operation, but the Ethereum *EIP process* that governs the final stage of turning a research proposal into something that actually makes it into the blockchain includes a fair share of vetocracy, though still less than Bitcoin. Governance over *irregular state changes*, hard forks that interfere with the operation of specific applications on-chain, is even more vetocratic: after the DAO fork, not a single proposal to intentionally "fix" some application by altering its code or moving its balance has been successful.

The case for vetocracy in these contexts is clear: it gives people a feeling of safety that the platform they build or invest on is not going to suddenly change the rules on them one day and destroy everything they've put years of their time or money into. Cryptocurrency proponents often cite Citadel interfering in Gamestop trading as an example of the opaque, centralized (and bulldozery) manipulation that they are fighting against. Web2 developers often complain about centralized platforms suddenly changing their APIs in ways that destroy startups built around their platforms. And, of course....

Vitalik Buterin, bulldozer victim

Ok fine, the story that WoW removing Siphon Life was the direct inspiration to Ethereum is exaggerated, but the infamous patch that ruined my beloved warlock and my response to it were very real!

And similarly, the case for vetocracy in politics is clear: it's a response to the often ruinous excesses of the bulldozers, both relatively minor and unthinkably severe, of the early 20th century.

So what's the synthesis?

The primary purpose of this point is to outline an axis, not to argue for a particular position. And if the vetocracy vs bulldozer axis is anything like the libertarian vs authoritarian axis, it's inevitably going to have internal subtleties and contradictions: much like a free society will see people voluntarily joining internally autocratic corporations (yes, even lots of people who are totally not economically desperate make such choices), many movements will be vetocratic internally but bulldozery in their relationship with the outside world.

But here are a few possible things that one could believe about bulldozers and vetocracy:

- The physical world has too much vetocracy, but the digital world has too many bulldozers, and there are no digital places that are truly effective refuges from the bulldozers (hence: why we need blockchains?)
- Processes that create durable change need to be bulldozery toward the status quo but protecting that change requires a vetocracy. There's some optimal rate at which such processes should happen; too much and there's chaos, not enough and there's stagnation.
- A few key institutions should be protected by strong vetocracy, and these institutions exist both to enable bulldozers needed to enact positive change *and* to give people things they can depend on that are not going to be brought down by bulldozers.
- In particular, blockchain base layers should be vetocratic, but application-layer governance should leave more space for bulldozers
- Better economic mechanisms ([quadratic voting](#)? [Harberger taxes](#)?) can get us many of the benefits of both vetocracy and bulldozers without many of the costs.

Vetocracy vs bulldozer is a particularly useful axis to use when thinking about *non-governmental* forms of human organization, whether for-profit companies, non-profit organizations, blockchains, or something else entirely. The relatively easier ability to exit from such systems (compared to governments) confounds discussion of how libertarian vs authoritarian they are, and so far blockchains and even centralized tech platforms have not really found many ways to differentiate themselves on the left vs right axis (though I would love to see more attempts at left-leaning crypto projects!). The vetocracy vs bulldozer axis, on the other hand, continues to map to non-governmental structures quite well - potentially making it very relevant in discussing these new kinds of non-governmental structures that are becoming increasingly important.

Endgame

2021 Dec 06

Special thanks to a whole bunch of people from Optimism and Flashbots for discussion and thought that went into this piece, and Karl Floersch, Phil Daian, Hasu and Alex Obadia for feedback and review.

Consider the average "big block chain" - very high block frequency, very high block size, many thousands of transactions per second, but also highly centralized: because the blocks are so big, only a few dozen or few hundred nodes can afford to run a fully participating node that can create blocks or verify the existing chain. What would it take to make such a chain acceptably trustless and censorship resistant, at least by my standards?

Here is a plausible roadmap:

- Add a **second tier of staking, with low resource requirements, to do distributed block validation**. The transactions in a block are split into 100 buckets, with a Merkle or Verkle tree state root after each bucket. Each second-tier staker gets randomly assigned to one of the buckets. A block is only accepted when at least 2/3 of the validators assigned to each bucket sign off on it.
- **Introduce either fraud proofs or ZK-SNARKs to let users directly (and cheaply) check block validity**. ZK-SNARKs can cryptographically prove block validity directly; fraud proofs are a simpler scheme where if a block has an invalid bucket, anyone can broadcast a fraud proof of just that bucket. This provides another layer of security on top of the randomly-assigned validators.
- **Introduce data availability sampling to let users check block *availability***. By using DAS checks, light clients can verify that a block was published by only downloading a few randomly selected pieces.
- **Add secondary transaction channels to prevent censorship**. One way to do this is to allow secondary stakers to submit lists of transactions which the *next main block must include*.
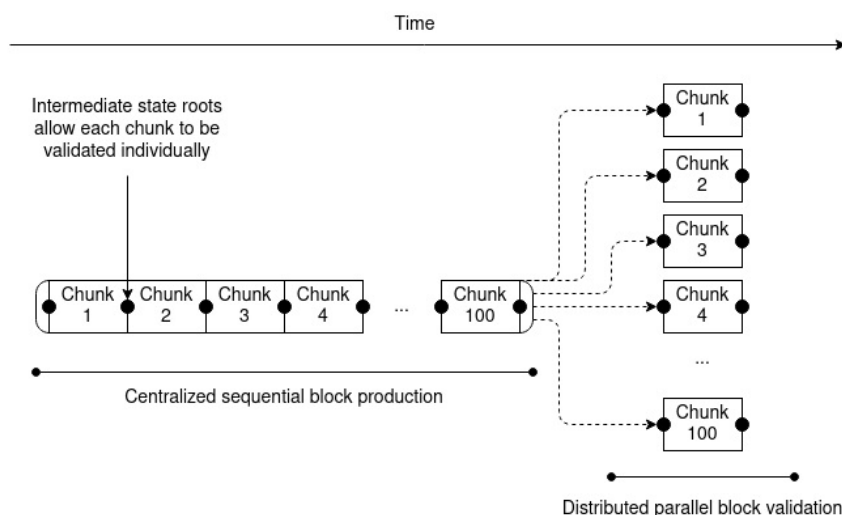
What do we get after all of this is done? **We get a chain where block *production* is still centralized, but block *validation* is trustless and highly decentralized, and specialized anti-censorship magic prevents the block producers from censoring.** It's somewhat aesthetically ugly, but it does provide the basic guarantees that we are looking for: even if every single one of the primary stakers (the block producers) is intent on attacking or censoring, the worst that they could do is all go offline entirely, at which point the chain stops accepting transactions until the community pools their resources and sets up *one* primary-staker node that is honest.

Now, consider one possible long-term future for rollups…

Imagine that one particular rollup - whether Arbitrum, Optimism, Zksync, StarkNet or something completely new - does a really good job of engineering their node implementation, to the point where it really can do 10,000 transactions per second if given powerful enough hardware. The techniques for doing this are in-principle well-known, and implementations were made by [Dan Larimer](#) and others many years ago: split up execution into one CPU thread running the unparallelizable but cheap business logic and a huge number of other threads running the expensive but highly parallelizable cryptography. Imagine also that Ethereum [implements sharding with data availability sampling](#), and has the space to store that rollup's on-chain data between its 64 shards. As a result, everyone migrates to this rollup. What would that world look like?
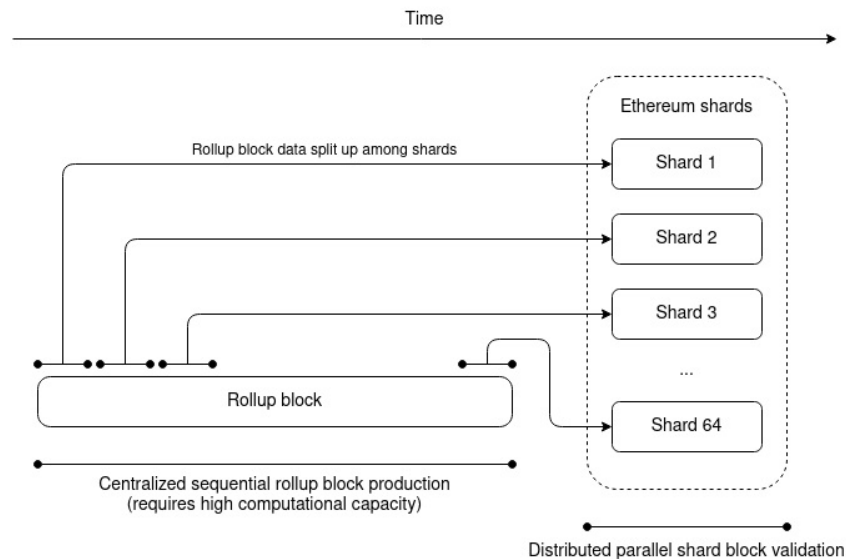

Once again, we get a world where, block *production* is centralized, block *validation* is trustless and highly decentralized, and censorship is still prevented. Rollup block producers have to process a huge number of transactions, and so it is a difficult market to enter, but they have no way to push invalid blocks through. Block availability is secured by the underlying chain, and block validity is guaranteed by the rollup logic: if it's a ZK rollup, it's ensured by SNARKs, and an optimistic rollup is secure as long as there is one honest actor somewhere running a fraud prover node (they can be subsidized with [Gitcoin grants](#)). Furthermore, because users always have the option of submitting transactions through the on-chain secondary inclusion channel, rollup sequencers also cannot effectively censor.

Now, consider the other possible long-term future of rollups…

No single rollup succeeds at holding anywhere close to the majority of Ethereum activity. Instead, they all top out at a few hundred transactions per second. We get a multi-rollup future for Ethereum - the [Cosmos multi–chain vision](#), but on top of a base layer providing data availability and shared security. Users frequently rely on [cross-rollup bridging](#) to jump between different rollups without paying the high fees on the main chain. What would that world look like?

It seems like we could have it all: decentralized validation, robust censorship resistance, and even distributed block *production*, because the rollups are all invididually small and so easy to start producing blocks in. But the decentralization of block production may not last, because of the possibility of [cross-domain MEV](#). There are a number of benefits to being able to construct the next block on *many domains at the same time*: you can create blocks that use arbitrage opportunities that rely on making transactions in two rollups, or one rollup and the main chain, or even more complex combinations.
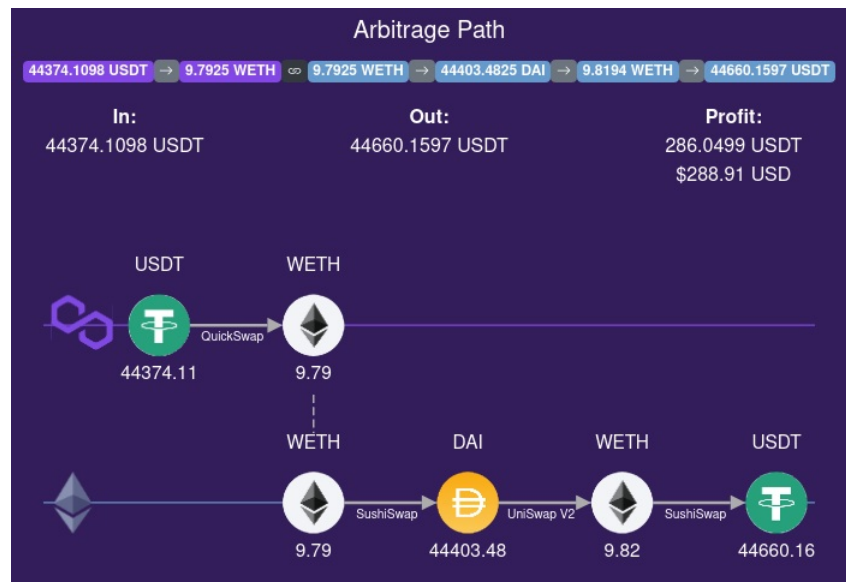
A cross-domain MEV opportunity discovered by [Western Gate](#)

Hence, in a multi-domain world, there are strong pressures toward the same people controlling block production on all domains. It may not happen, but there's a good chance that it will, and we have to be prepared for that possibility. What can we do about it? So far, the best that we know how to do is to use two techniques in combination:

- Rollups implement some mechanism for auctioning off block production at each slot, or the Ethereum base layer implements **[proposer/builder separation (PBS)](#)** (or both). This ensures that at least any centralization tendencies in block production don't lead to a completely elite-captured and concentrated staking pool market dominating block validation.
- Rollups implement **censorship-resistant bypass channels**, and the Ethereum base layer implements [PBS anti-censorship techniques](#). This ensures that if the winners of the potentially highly centralized "pure" block production market try to censor transactions, there are ways to bypass the censorship.

So what's the result? **Block *production* is centralized, block *validation* is trustless and highly decentralized, and censorship is still prevented**.
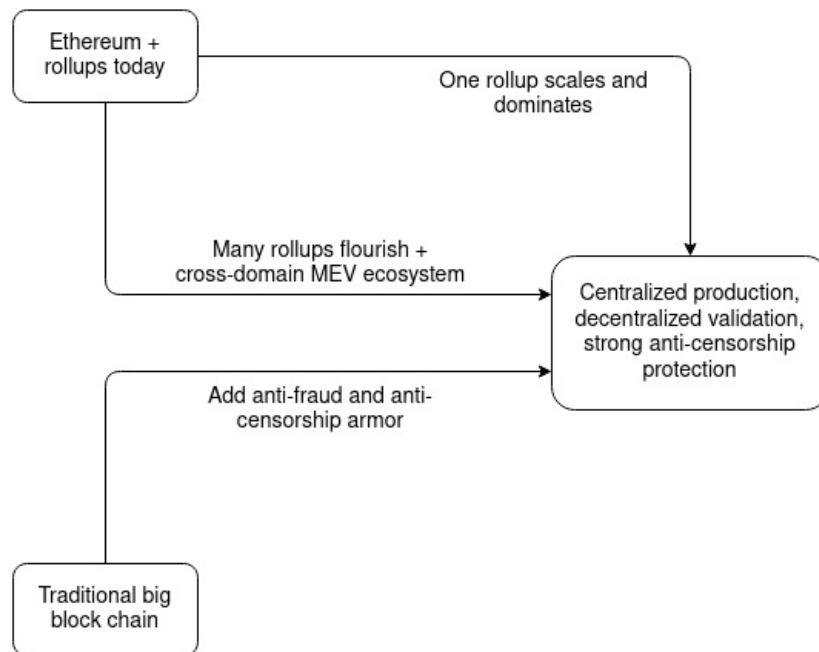
Three paths toward the same destination.

So what does this mean?

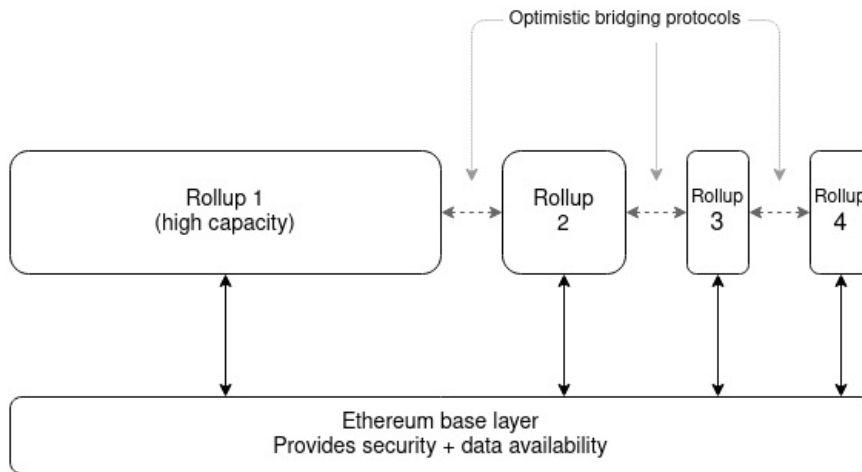While there are many paths toward building a scalable and secure long-term blockchain ecosystem, it's looking like they are all building toward very similar futures. There's a high chance that block production will end up centralized: either the network effects within rollups or the network effects of cross-domain MEV push us in that direction in their own different ways. But what we *can* do is use protocol-level techniques such as committee validation, data availability sampling and bypass channels to "regulate" this market, ensuring that the winners cannot abuse their power.

What does this mean for block producers? Block production is likely to become a specialized market, and the domain expertise is likely to carry over across different domains. 90% of what makes a good Optimism block producer also makes a good Arbitrum block producer, and a good Polygon block producer, and even a good Ethereum base layer block producer. If there are many domains, cross-domain arbitrage may also become an important source of revenue.

What does this mean for Ethereum? First of all, Ethereum is very well-positioned to adjust to this future world, despite the inherent uncertainty. The profound benefit of the Ethereum rollup-centric roadmap is that it means that Ethereum is open to all of the futures, and does not have to commit to an opinion about which one will necessarily win. Will users very strongly want to be on a single rollup? Ethereum, following its existing course, can be the base layer of that, automatically providing the anti-fraud and anti-censorship "armor" that high-capacity domains need to be secure. Is making a high-capacity domain too technically complicated, or do users just have a great need for variety? Ethereum can be the base layer of that too - and a very good one, as the common root of trust makes it far easier to move assets between rollups safely and cheaply.

But also, Ethereum researchers should think hard about what levels of decentralization in block production are actually achievable. It may not be worth it to add complicated plumbing to make highly decentralized block production easy if cross-domain MEV (or even cross-shard MEV from one rollup taking up multiple shards) make it unsustainable regardless.

Optimistic bridging protocols

Rollup 1 (high capacity)

Rollup 2

Rollup 3

Rollup 4

Ethereum base layer
Provides security + data availability

What does this mean for big block chains? There is a path for them to turn into something trustless and censorship resistant, and we'll soon find out if their core developers and communities actually value censorship resistance and decentralization enough for them to do it!

It will likely take years for all of this to play out. Sharding and data availability sampling are complex technologies to implement. It will take years of refinement and audits for people to be fully comfortable storing their assets in a ZK-rollup running a full EVM. And cross-domain MEV research too is still in its infancy. But it does look increasingly clear how a realistic but bright future for scalable blockchains is likely to emerge.

Review of Optimism retro funding round 1

2021 Nov 16

Special thanks to Karl Floersch and Haonan Li for feedback and review, and Jinglan Wang for discussion.

Last month, Optimism ran their first round of retroactive public goods funding, allocating a total of $1 million to 58 projects to reward the good work that these projects have already done for the Optimism and Ethereum ecosystems. In addition to being the first major retroactive general-purpose public goods funding experiment, it's also the first experiment in a new kind of governance through **badge holders** - not a very small decision-making board and also not a fully public vote, but instead a quadratic vote among a medium-sized group of 22 participants.

The entire process was highly transparent from start to finish:

- The rules that the badge holders were supposed to follow were enshrined in the badge holder instructions
- You can see the projects that were nominated in this spreadsheet
- All discussion between the badge holders happened in publicly viewable forums. In addition to Twitter conversation (eg. Jeff Coleman's thread and also others), all of the explicit structured discussion channels were publicly viewable: the #retroactive-public-goods channel on the Optimism discord, and a published Zoom call
- The full results, and the individual badge holder votes that went into the results, can be viewed in this spreadsheet

And finally, here are the results in an easy-to-read chart form:

Ethersjs	$51,345
go-ethereum	$45,232
EthGlobal	$42,787
Hardhat	$41,565
WalletConnect	$40,342
Solidity	$40,342
Goerli	$35,452
Erigon	$34,230
Hive	$31,785
DappTools	$30,562
BuidlGuidl	$28,117
solidity-coverage	$24,450
Kovan	$24,450
Etherscan	$24,450
Giveth	$24,450
blst - BLS12-381 Signatures (Supranational)	$23,227
CryptoFees.info	$22,005
Rotki	$22,005
Proof of Humanity	$22,005
Ipsilon	$20,782
Prettier solidity (library for IDEs)	$20,782
js-ethereum-cryptography (JS library)	$19,560
Frame	$18,337
OVM security research	$17,115
hardhat-deploy	$17,115

Much like the Gitcoin quadratic funding rounds and the MolochDAO grants, this is yet another instance of the Ethereum ecosystem establishing itself as a key player in the innovative public goods funding mechanism design space. But what can we learn from this experiment?

Analyzing the results

First, let us see if there are any interesting takeaways that can be seen by looking at the results. But what do we compare the results to? The most natural point of comparison is the other major public goods funding experiment that we've had so far: the Gitcoin quadratic funding rounds (in this case, round 11).

Gitcoin round 11 (tech only)		Optimism retro round 1	
Li.Finance - Cross-Chain Bridge Aggregator	$32,165	Ethersjs	$51,345
Nym	$30,914	go-ethereum	$45,232
Dark Forest	$29,206	EthGlobal	$42,787
Brownie	$25,476	Hardhat	$41,565
MatrixETF - The Next Generation of ETF	$24,932	WalletConnect	$40,342
BuidlGuidl	$24,641	Solidity	$40,342
Frame: Privacy Focused Native Ethereum W	$22,913	Goerli	$35,452
Otterscan	$20,006	Erigon	$34,230
Connext Network	$15,821	Hive	$31,785
Umbra: Privacy Preserving Stealth Payments	$14,546	DappTools	$30,562
ArchiveNode.io - The Public Access Ether	$14,098	BuidlGuidl	$28,117
Uniswap V3 Options	$12,696	solidity-coverage	$24,450
RSS3 - RSS with human curation	$12,032	Kovan	$24,450
clr.fund	$12,025	Etherscan	$24,450
Llama - Treasury Management for DAOs	$11,752	Giveth	$24,450
Revert	$10,954	blst - BLS12-381 Signatures (Supranational)	$23,227
Rotki - The portfolio tracker and accounting	$10,368	CryptoFees.info	$22,005
Defi SDK Polywrapper	$9,644	Rotki	$22,005
Atlantis World	$9,420	Proof of Humanity	$22,005
ZeroPool - Scaling anonymous transactions	$8,315	Ipsilon	$20,782
Prysm by Prysmatic Labs	$7,591	Prettier solidity (library for IDEs)	$20,782
TypeChain	$7,520	js-ethereum-cryptography (JS library)	$19,560
BrightID ☀ Universal Proof of Uniqueness	$6,862	Frame	$18,337
Proof of Humanity	$6,444	OVM security research	$17,115
zkpay	$6,366	hardhat-deploy	$17,115

Probably the most obvious property of the Optimism retro results that can be seen without any comparisons is the category of the winners: **every major Optimism**

retro winner was a technology project. There was nothing in the badge holder instructions that specified this; non-tech projects (say, the translations at ethereum.cn) were absolutely eligible. And yet, due to some combination of choice of badge holders and subconscious biases, the round seems to have been understood as being tech-oriented. Hence, I restricted the Gitcoin results in the table above to technology ("DApp Tech" + "Infra Tech") to focus on the remaining differences.

Some other key remaining differences are:

- **The retro round was low variance**: the top-receiving project only got three times more (in fact, *exactly* three times more) than the 25th, whereas in the Gitcoin chart combining the two categories the gap was over 5x, and if you look at DApp Tech or Infra Tech separately the gap is *over 15*x! I personally blame this on Gitcoin using standard quadratic funding (\(reward \approx (\sum_i \sqrt x_i) ^2\)) and the retro round using \(\sum_i \sqrt x_i\) without the square; perhaps the next retro round should just add the square.
- **The retro round winners are more well-known projects**: this is actually an *intended* consequence: the retro round focused on rewarding projects for value already provided, whereas the Gitcoin round was open-ended and many contributions were to promising new projects in expectation of future value.
- **The retro round focused more on infrastructure, the Gitcoin round more on more user-facing projects**: this is of course a generalization, as there are plenty of infrastructure projects in the Gitcoin list, but *in general* applications that are directly user-facing are much more prominent there. A particularly interesting consequence (or cause?) of this is that the Gitcoin round more on projects appealing to sub-communities (eg. gamers), whereas the retro round focused more on globally-valuable projects - or, less charitably, projects appealing to the one particular sub-community that is Ethereum developers.

It is my own (admittedly highly subjective) opinion that the retro round winner selection is somewhat higher quality. This is independent of the above three differences; it's more a general impression that the specific projects that were chosen as top recipients on the right were very high quality projects, to a greater extent than top recipients on the left.

Of course, this could have two causes: (i) a smaller but more skilled number of badge holders ("technocrats") can make better decisions than "the crowd", and (ii) it's easier to judge quality retroactively than ahead of time. And this gets us an interesting question: **what if a simple way to summarize much of the above findings is that technocrats are smarter but the crowd is more diverse**?

Could we make badge holders and their outputs more diverse?

To better understand the problem, let us zoom in on the one specific example that I already mentioned above: ethereum.cn. This is an excellent Chinese Ethereum community project (though not the only one! See also EthPlanet), which has been providing a lot of resources in Chinese for people to learn about Ethereum, including translations of many highly technical articles written by Ethereum community members and about Ethereum originally in English.

Ethereum.cn webpage. Plenty of high quality technical material - though they have still not yet gotten the memo that they were supposed to rename "eth2" to "consensus layer". Minus ten retroactive reward points for them.

What knowledge does a badge holder need to be able to effectively determine whether ethereum.cn is an awesome project, a well-meaning but mediocre project that few Chinese people actually visit, or a scam? Likely the following:

- Ability to speak and understand Chinese
- Being plugged into the Chinese community and understanding the social dynamics of that specific project
- Enough understanding of both the tech *and* of the frame of mind of non-technical readers to judge the site's usefulness for them

Out of the current, heavily US-focused, badge holders, the number that satisfy these requirements is basically zero. Even the two Chinese-speaking badge holders are US-based and not close to the Chinese Ethereum community.

So, what happens if we expand the badge holder set? We could add five badge holders from the Chinese Ethereum community, five from India, five from Latin America, five from Africa, and five from Antarctica to represent the penguins. At the same time we could also diversify among areas of expertise: some technical experts, some community leaders, some people plugged into the Ethereum gaming world. Hopefully, we can get enough coverage that for each project that's valuable to Ethereum, we would have at least 1-5 badge holders who understands enough about that project to be able to intelligently vote on it. But then we see the problem: there would *only* be 1-5 badge holders able to intelligently vote on it.

There are a few families of solutions that I see:

1. **Tweak the quadratic voting design**. In theory, quadratic voting has the unique property that there's very little incentive to vote on projects that you do not understand. Any vote you make takes away credits that you could use to vote on projects you understand better. However, the current quadratic voting design has a flaw here: not voting on a project isn't truly a neutral non-vote, it's a vote for the project getting nothing. I don't yet have great ideas for how to do this. A key question is: if zero becomes a truly neutral vote, then how much money does a project get if nobody makes *any* votes on it? However, this is worth looking into more.
2. **Split up the voting into categories or sub-committees**. Badge holders would first vote to sort projects into buckets, and the badge holders within each bucket ("zero knowledge proofs", "games", "India"...) would then make the decisions from there. This could also be done in a more "liquid" way through delegation - a badge holder could select some other badge holder to decide their vote on some project, and they would automatically copy their vote.
3. **Everyone still votes on everything, but facilitate more discussion**. The badge holders that *do* have the needed domain expertise to assess a given project (or a single aspect of some given project) come up with their opinions and write a document or spreadsheet entry to express their reasoning. Other badge holders use this information to help make up their minds.

Once the number of decisions to be made gets even higher, we could even consider ideas like **in-protocol random sortition** (eg. see this idea to incorporate sortition into quadratic voting) to reduce the number of decisions that each participant needs to make. Quadratic sortition has the particularly nice benefit that it naturally leads to large decisions being made by the entire group and small decisions being made by smaller groups.

The means-testing debate

In the post-round retrospective discussion among the badgeholders, one of the key questions that was brought up is: when choosing which projects to fund, **should badge holders take into account whether that project is still in dire need of funding**, and de-prioritize projects that are already well-funded through some other means? That is to say, should retroactive rewards be [means-tested](#)?

In a "regular" grants-funding round, the rationale for answering "yes" is clear: increasing a project's funding from $0 to $100k has a much bigger impact on its ability to do its job than increasing a project's funding from $10m to $10.1m. But Optimism retro funding round 1 is not a regular grants-funding round. **In retro funding, the objective is *not* to give people money in expectation of future work that money could help them do. Rather, the objective is to reward people for work already done, to change the incentives for anyone working on projects *in the future*.** With this in mind, to what extent should *retroactive* project funding depend on how much a given project actually needs the funds?

The case for means testing

Suppose that you are a 20-year-old developer, and you are deciding whether to join some well-funded defi project with a fancy token, or to work on some cool open-source fully public good work that will benefit everyone. If you join the well-funded defi project, you will get a $100k salary, and your financial situation will be guaranteed to be very secure. If you work on public-good projects on your own, you will have no income. You have some savings and you could make some money with side gigs, but it will be difficult, and you're not sure if the sacrifice is worth it.

Now, consider two worlds, **World A** and **World B**. First, the similarities:

- There are ten people exactly like you out there that could get retro rewards, and five of you will. Hence, there's a **50% chance you'll get a retro reward**.
- Theres's a **30% chance that your work will propel you to moderate fame** and you'll be hired by some company with even better terms than the original defi project (or even start your own).

Now, the differences:

- **World A (means testing)**: retro rewards are concentrated among the actors that do *not* find success some other way
- **World B (no means testing)**: retro rewards are given out independently of whether or not the project finds success in some other way

Let's look at your chances in each world.

Event	Probability (World A)	Probability (World B)
Independent success *and* retroactive reward	0%	15%
Independent success only	30%	15%
Retroactive reward only	50%	35%
Nothing	20%	35%

From your point of view as a non-risk-neutral human being, the 15% chance of getting success twice in world B matters much less than the fact that in world B your chances of being left completely in the cold with nothing are nearly double.

Hence, if we want to encourage people in this hypothetical 20 year old's position to actually contribute, concentrating retro rewards among projects who did not already get rewarded some other way seems prudent.

The case against means testing

Suppose that you are someone who contributes a small amount to many projects, or an investor seed-funding public good projects in anticipation of retroactive rewards. In this case, the share that you would get from any single retroactive reward is small. Would you rather have a 10% chance of getting $10,100, or a 10% chance of getting $10,000 and a 10% chance of getting $100? It really doesn't matter.

Furthermore, your chance of getting rewarded via retroactive funding may well be quite disjoint from your chance of getting rewarded some other way. There are countless stories on the internet of people putting a big part of their lives into a project when that project was non-profit and open-source, seeing that project go for-profit and become successful, and getting absolutely nothing out of it for themselves. In all of these cases, it doesn't really matter whether or not retroactive rewards care on whether or not projects are needy. In fact, it would probably be *better* for them to just focus on judging quality.

Means testing has downsides of its own. It would require badge holders to expend effort to determine to what extent a project is well-funded outside the retroactive reward system. It could lead to projects expending effort to hide their wealth and *appear* scrappy to increase their chance of getting more rewards. Subjective evaluations of neediness of recipients could turn into politicized evaluations of moral worthiness of recipients that introduce more controversy into the mechanism. In the extreme, an elaborate tax return system might be required to properly enforce fairness.

What do I think?

In general, it seems like doing a little bit of prioritizing projects that have not discovered business models has advantages, but we should not do too much of that. Projects should be judged by their effect on the world first and foremost.

Nominations

In this round, anyone could nominate projects by submitting them in a Google form, and there were only 106 projects nominated. What about the next round, now that people know for sure that they stand a chance at getting thousands of dollars in payout? What about the round a year in the hypothetical future, when fees from millions of daily transactions are paying into the retro funding rounds, and individual projects are getting more money than the entire round is today?

Some kind of multi-level structure for nominations seems inevitable. There's probably no need to enshrine it directly into the voting rules. Instead, we can look at this as one particular way of changing the structure of the discussion: nomination rules filter out the nominations that badge holders need to look at, and anything the badge holders do not look at will get zero votes by default (unless a badge holder *really* cares to bypass the rules because they have their own reasons to believe that some project is valuable).

Retro funding today   Retro funding tomorrow
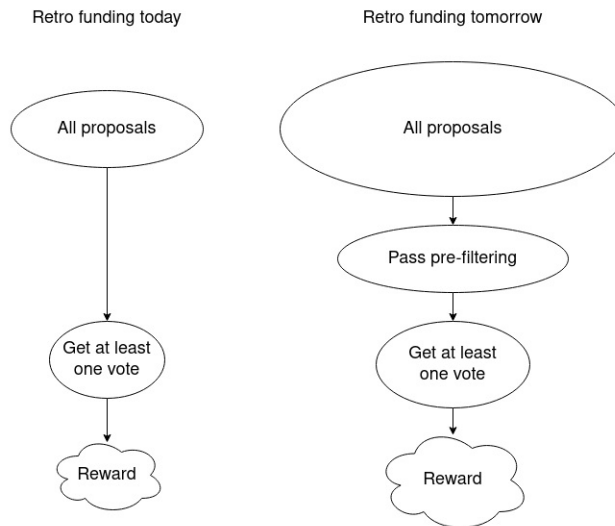
Some possible ideas:

- **Badge holder pre-approval**: for a proposal to become visible, it must be approved by N badge holders (eg. N=3?). Any N badge holders could pre-approve any project; this is an anti-spam speed bump, not a gate-keeping sub-committee.
- **Require proposers to provide more information** about their proposal, justifying it and reducing the work badge holders need to do to go through it. Badge holders would also appoint a separate committee and entrust it with sorting through these proposals and forwarding the ones that follow the rules and pass a basic smell test of not being spam
- **Proposals have to specify a category** (eg. "zero knowledge proofs", "games", "India"), and badge holders who had declared themselves experts in that category would review those proposals and forward them to a vote only if they chose the right category and pass a basic smell test.
- **Proposing requires a deposit** of 0.02 ETH. If your proposal gets 0 votes (alternatively: if your proposal is explicitly deemed to be "spam"), your deposit is lost.
- **Proposing requires a proof-of-humanity ID**, with a maximum of 3 proposals per human. If your proposals get 0 votes (alternatively: if any of your proposals is explicitly deemed to be "spam"), you can no longer submit proposals for a year (or you have to provide a deposit).

Conflict of interest rules

The first part of the post-round retrospective discussion was taken up by discussion of conflict of interest rules. The badge holder instructions include the following lovely clause:

> 6. **No self-dealing or conflicts of interest**
> RetroDAO governance participants should refrain from voting on sending funds to organizations where any portion of those funds is expected to flow to them, their other projects, or anyone they have a close personal or economic relationship with.

As far as I can tell, this was honored. Badge holders did not try to self-deal, as they were (as far as I can tell) good people, and they knew their reputations were on the line. But there were also some subjective edge cases:

- **Wording issues causing confusion**. Some badge holders wondered about the word "other": could badge holders direct funds to their *own* primary projects? Additionally, the "sending funds to organizations where..." language does not strictly prohibit *direct* transfers to self. These were arguably simple mistakes in writing this clause; the word "other" should just be removed and "organizations" replaced with "addresses".
- **What if a badge holder is part of a nonprofit** that itself gives out grants to other projects? Could the badge holder vote for that nonprofit? The badge holder would not *benefit*, as the funds would 100% pass-through to others, but they could benefit indirectly.
- **What level of connection counts as close connection**? Ethereum is a tight-knit community and the people qualified to judge the best projects are often at least to some degree friends with the team or personally involved in those projects precisely because they respect those projects. When do those connections step over the line?

I don't think there are perfect answers to this; rather, the line will inevitably be gray and can only be discussed and refined over time. The main mechanism-design tweaks that can mitigate it are (i) increasing the number of badge holders, diluting the portion of them that can be insiders in any single project, (ii) reducing the rewards going to projects that *only* a few badge holders support (my suggestion above to set the reward to $(\sum_i \sqrt x_i)^2$ instead of $\sum_i \sqrt x_i$ would help here too), and (iii) making sure it's possible for badge holders to counteract clear abuses if they do show up.

Should badge holder votes be secret ballot?

In this round, badge holder votes were completely transparent; anyone can see how each badge holder votes. But transparent voting has a huge downside: it's vulnerable to bribery, including informal bribery of the kind that even good people easily succumb to. Badge holders could end up supporting projects in part with the subconscious motivation of winning favor with them. Even more realistically, badge holders may be unwilling to make *negative* votes even when they are justified, because a public negative vote could easily rupture a relationship.

Secret ballots are the natural alternative. Secret ballots are used widely in democratic elections where any citizen (or sometimes resident) can vote, precisely to prevent vote buying and more coercive forms of influencing how people vote. However, in typical elections, **votes within executive and legislative bodies are typically public**. The usual reasons for this have to do with theories of democratic accountability: voters need to know how their representatives vote so that they can choose their representatives and know that they are not completely lying about their stated values. But there's also a dark side to accountability: elected officials making public votes are accountable to anyone who is trying to bribe them.

Secret ballots within government bodies do have precedent:

- The Israeli Knesset uses secret votes to elect the president and a few other officials
- The Italian parliament has used secret votes in a variety of contexts. In the 19th century, it was considered an important way to protect parliament votes from interference by a monarchy.
- Discussions in US parliaments were less transparent before 1970, and some researchers argue that the switch to more transparency led to more corruption.
- Voting in juries is often secret. Sometimes, even the identities of jurors are secret.

In general, the conclusion seems to be that secret votes in government bodies have complicated consequences; it's not clear that they should be used everywhere, but it's also not clear that transparency is an absolute good either.

In the context of Optimism retro funding specifically, the main specific argument I heard against secret voting is that it would make it harder for badge holders to rally and vote against other badge holders making votes that are clearly very wrong or even malicious. Today, if a few rogue badge holders start supporting a project that has not provided value and is clearly a cash grab for those badge holders, the other badge holders can see this and make negative votes to counteract this attack. With secret ballots, it's not clear how this could be done.

I personally would favor the second round of Optimism retro funding using completely secret votes (except perhaps open to a few researchers under conditions of non-disclosure) so we can tell what the material differences are in the outcome. Given the current small and tight-knit set of badge holders, dealing with rogue badge hodlers is likely not a primary concern, but in the future it will be; hence, coming up with a secret ballot design that allows counter-voting or some alternative strategy is an important research problem.

Other ideas for structuring discussion

The level of participation among badge holders was very uneven. Some (particularly Jeff Coleman and Matt Garnett) put a lot of effort into their participation, publicly expressing their detailed reasoning in Twitter threads and helping to set up calls for more detailed discussion. Others participated in the discussion on Discord and still others just voted and did little else.

There was a choice made (ok fine, I was the one who suggested it) that the #retroactive-public-goods channel should be readable by all (it's in the Optimism discord), but to prevent spam only badge holders should be able to speak. This reduced many people's ability to participate, especially ironically enough my own (I am not a badge holder, and my self-imposed Twitter quarantine, which only allows me to tweet links to my own long-form content, prevented me from engaging on Twitter).

These two factors together meant that there was not that much discussion taking place; certainly less than I had been hoping for. What are some ways to encourage more discussion?

Some ideas:

- **Badge holders could vote in advisors**, who cannot vote but can speak in the #retroactive-public-goods channel and other badge-holder-only meetings.
- **Badge holders could be required to explain their decisions**, eg. writing a post or a paragraph for each project they made votes on.
- **Consider compensating badge holders**, either through an explicit fixed fee or through a norm that badge holders themselves who made exceptional contributions to discussion are eligible for rewards in future rounds.
- **Add more discussion formats**. If the number of badge holders increases and there are subgroups with different specialties, there could be more chat rooms and each of them could invite outsiders. Another option is to create a dedicated subreddit.

It's probably a good idea to start experimenting with more ideas like this.

Conclusions

Generally, I think Round 1 of Optimism retro funding has been a success. Many interesting and valuable projects were funded, there was quite a bit of discussion, and all of this despite it only being the first round.

There are a number of ideas that could be introduced or experimented with in subsequent rounds:

- **Increase the number and diversity of badge holders**, while making sure that there is *some* solution to the problem that only a few badge holders will be experts in any individual project's domain.
- **Add some kind of two-layer nomination structure**, to lower the decision-making burden that the entire badge holder set is exposed to
- **Use secret ballots**
- **Add more discussion channels, and more ways for non-badge-holders to participate**. This could involve reforming how existing channels work, or it could involve adding new channels, or even specialized channels for specific categories of projects.
- **Change the reward formula to increase variance**, from the current $\sum_i \sqrt x_i$ to the standard quadratic funding formula of $(\sum_i \sqrt x_i)^2$.

In the long term, if we want retro funding to be a sustainable institution, there is also the question of **how new badge holders are to be chosen** (and, in cases of malfeasance, **how badge holders could be removed**). Currently, the selection is centralized. In the future, we need some alternative. One possible idea for round 2 is to simply allow existing badge holders to vote in a few new badge holders. In the longer term, to prevent it from being an insular bureaucracy, perhaps one badge holder each round could be chosen by something with more open participation, like a proof-of-humanity vote.

In any case, retroactive public goods funding is still an exciting and new experiment in institutional innovation in multiple ways. It's an experiment in non-coin-driven decentralized governance, and it's an experiment in making things happen through retroactive, rather than proactive, incentives. To make the experiment fully work, a lot more innovation will need to happen both in the mechanism itself and in the ecosystem that needs to form around it. When will we see the first retro funding-focused angel investor? Whatever ends up happening, I'm looking forward to seeing how this experiment evolves in the rounds to come.

Halo and more: exploring incremental verification and SNARKs without pairings

2021 Nov 05

Special thanks to Justin Drake and Sean Bowe for wonderfully pedantic and thoughtful feedback and review, and to Pratyush Mishra for discussion that contributed to the original IPA exposition.

Readers who have been [following](#) the [ZK-SNARK](#) space [closely](#) should by now be familiar with the high level of how ZK-SNARKs work. ZK-SNARKs are based on checking equations where the elements going into the equations are mathematical abstractions like [polynomials](#) (or in [rank-1 constraint systems](#) matrices and vectors) that can hold a lot of data. There are three major families of cryptographic technologies that allow us to represent these abstractions succinctly: Merkle trees (for [FRI](#)), regular elliptic curves (for [inner product arguments (IPAs))](#), and elliptic curves with pairings and trusted setups (for [KZG commitments](#)). These three technologies lead to the three types of proofs: FRI leads to STARKs, KZG commitments lead to "regular" SNARKs, and IPA-based schemes lead to bulletproofs. These three technologies have very distinct tradeoffs:

Technology	Cryptographic assumptions	Proof size	Verification time
FRI	Hashes only (quantum safe!)	Large (10-200 kB)	Medium (poly-logarithmic)
Inner product arguments (IPAs)	Basic elliptic curves	Medium (1-3 kB)	**Very high (linear)**
KZG commitments	Elliptic curves + pairings + trusted setup	Short (~500 bytes)	Low (constant)

So far, the first and the third have seen the most attention. The reason for this has to do with that pesky right column in the second row of the table: elliptic curve-based inner product arguments have linear verification time. What this means that even though the *size* of a proof is small, the amount of time needed to verify the proof always takes longer than just running the computation yourself. This makes IPAs non-viable for scalability-related ZK-SNARK use cases: there's no point in using an IPA-based argument to prove the validity of an Ethereum block, because verifying the proof will take longer than just checking the block yourself. KZG and FRI-based proofs, on the other hand, really are much faster to verify than doing the computation yourself, so one of those two seems like the obvious choice.

More recently, however, there has been a slew of research into techniques for *merging* multiple IPA proofs into one. Much of the initial work on this was done as part of designing the [Halo protocol](#) which is [going into Zcash](#). These merging techniques are cheap, and a merged proof takes no longer to verify than a single one of the proofs that it's merging. This opens a way forward for IPAs to be useful: instead of verifying a size-n computation with a proof that takes still takes $O(n)$ time to verify, break that computation up into smaller size-k steps, make $\frac{n}{k}$ proofs for each step, and merge them together so the verifier's work goes down to a little more than $O(k)$. These techniques also allow us to do *incremental verification*: if new things keep being introduced that need to be proven, you can just keep taking the existing proof, mixing it in with a proof of the new statement, and getting a proof of the new combined statement out. This is really useful for verifying the integrity of, say, an entire blockchain.

So how do these techniques work, and what can they do? That's exactly what this post is about.

Background: how do inner product arguments work?

Inner product arguments are a proof scheme that can work over many mathematical structures, but usually we focus on IPAs over [elliptic curve points](#). IPAs can be made over simple elliptic curves, theoretically even Bitcoin and Ethereum's [secp256k1](#) (though some special properties are preferred

to make [FFTs](#) more efficient); no need for insanely complicated pairing schemes that despite having written an [explainer article](#) and an [implementation](#) I can still barely understand myself.

We'll start off with the commitment scheme, typically called **Pedersen vector commitments**. To be able to commit to degree $< n$ polynomials, we first publicly choose a set of base points, $G_0 ... G_{n-1}$. These points can be generated through a pseudo-random procedure that can be re-executed by anyone (eg. the x coordinate of G_i can be $hash(i, j)$ for the lowest integer $j \ge 0$ that produces a valid point); this is *not* a trusted setup as it does not rely on any specific party to introduce secret information.

To commit to a polynomial $P(x) = \sum_i c_i x^i$, the prover computes $com(P) = \sum_i c_i G_i$. For example, $com(x^2 + 4)$ would equal $G_2 + 4 * G_0$ (remember, the $+$ and $*$ here are [elliptic curve addition](#) and multiplication). Cryptographers will also often add an extra $r \cdot H$ hiding parameter for privacy, but for simplicity of exposition we'll ignore privacy for now; in general, it's not that hard to add privacy into all of these schemes.
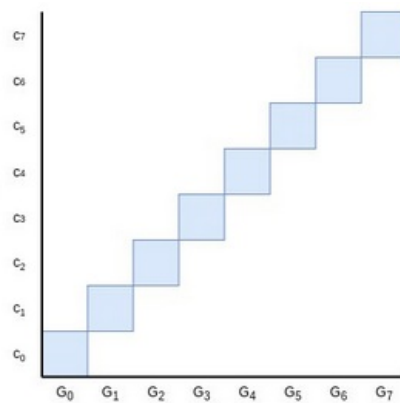

Though it's not really mathematically accurate to think of elliptic curve points as being like real numbers that have sizes, area is nevertheless a good intuition for thinking about linear combinations of elliptic curve points like we use in these commitments. The blue area here is the value of the Pedersen commitment $C = \sum_i c_i G_i$ to the polynomial $P = \sum_i c_i x^i$.

Now, let's get into how the proof works. **Our final goal will be a polynomial evaluation proof**: given some z, we want to make a proof that $P(z) = a$, where this proof can be verified by anyone who has the commitment $C = com(P)$. **But first, we'll focus on a simpler task: proving that C is a *valid commitment to any polynomial at all*** - that is, proving that C was constructed by taking a linear combination $\sum_i c_i G_i$ of the points $\{G_0 ... G_{n-1}\}$, without anything else mixed in.

Of course, technically *any* point is some multiple of G_0 and so it's theoretically a valid commitment of something, but what we care about is proving that the prover *knows* some $\{c_0 ... c_{n-1}\}$ such that $\sum_i c_i G_i = C$. A commitment C cannot commit to multiple distinct polynomials *that the prover knows about*, because if it could, that would imply that elliptic curves are broken.

The prover *could*, of course, just provide $\{c_0 ... c_{n-1}\}$ directly and let the verifier check the commitment. But this takes too much space. So instead, we try to reduce the problem to a smaller problem of half the size. The prover provides two points, L and R, representing the yellow and green areas in this diagram:
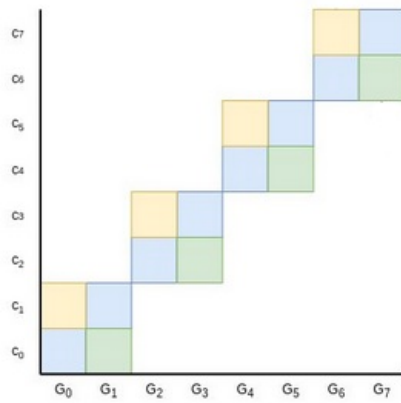
You may be able to see where this is going: if you add $C + L + R$ together (remember: C was the original commitment, so the blue area), the new combined point can be expressed as a sum of *four* squares instead of eight. And so now, the prover could finish by providing only four sums, the widths of each of the new squares. Repeat this protocol two more times, and we're down to a single full square, which the prover can prove by sending a single value representing its width.

But there's a problem: if C is incorrect in some way (eg. the prover added some extra point H into it), then the prover could just subtract H from L or R to compensate for it. We plug this hole by randomly scaling our points after the prover provides L and R:
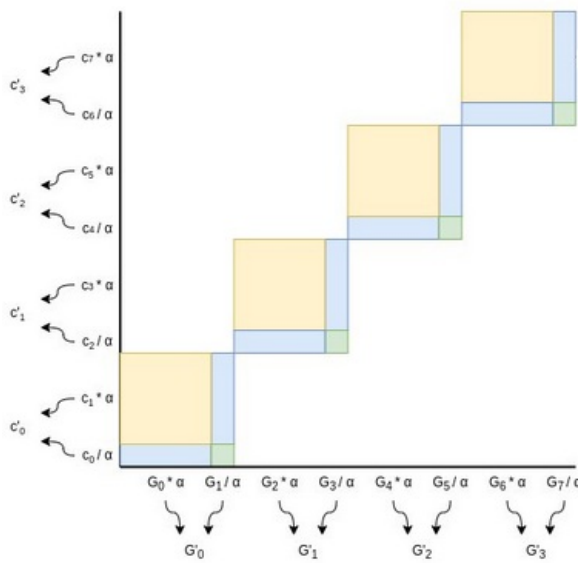
Choose a random factor α (typically, we set α to be the hash of all data added to the proof so far, including the L and R, to ensure the verifier can also compute α). Every even G_i point gets scaled by α, every odd G_i point gets scaled *down* by the same factor. Every odd coefficient gets scaled up by α (notice the flip), and every even coefficient gets scaled down by α. Now, notice that:

- The yellow area (L) gets multiplied by α^2 (because every yellow square is scaled up by α on both dimensions)
- The green area (R) gets divided by α^2 (because every green square is scaled down by α on both dimensions)
- The blue area (C) remains unchanged (because its width is scaled up but its height is scaled down)

Hence, we can generate our new half-size instance of the problem with some simple transformations:

- $G'_{i} = \alpha G_{2i} + \frac{G_{2i+1}}{\alpha}$

- $c'_{i} = \frac{c_{2i}}{\alpha} + \alpha c_{2i+1}$
- $C' = C + \alpha^2 L + \frac{R}{\alpha^2}$

(Note: in some implementations you instead do $G'_i = \alpha G_{2i} + G_{2i+1}$ and $c'_i = c_{2i} + \alpha c_{2i+1}$ without dividing the odd points by α. This makes the equation $C' = \alpha C + \alpha^2 L + R$, which is less symmetric, but ensures that the function to compute any G' in any round of the protocol becomes a polynomial without any division. [Yet *another* alternative](#) is to do $G'_i = \alpha G_{2i} + G_{2i+1}$ and $c'_i = c_{2i} + \frac{c_{2i+1}}{\alpha}$, which avoids any α^2 terms.)

And then we repeat the process until we get down to one point:
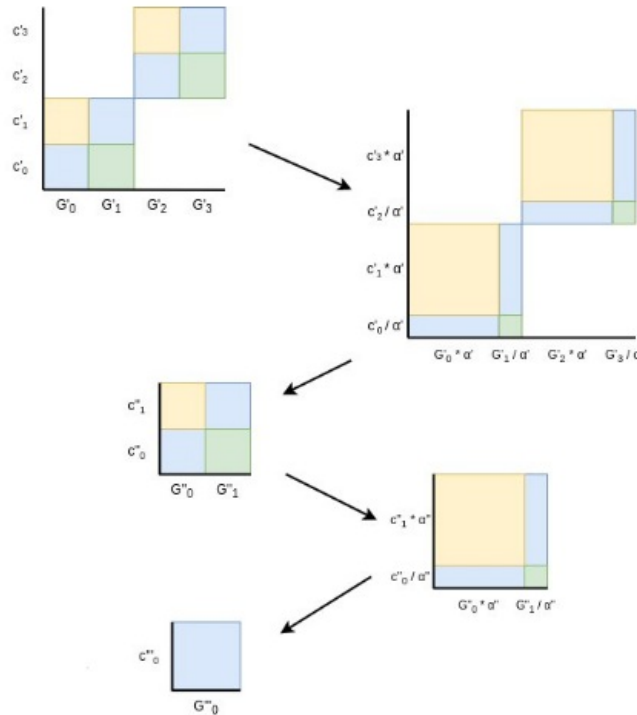
Finally, we have a size-1 problem: prove that the final modified C^* (in this diagram it's C''' because we had to do three iterations, but it's $\log(n)$ iterations generally) equals the final modified G^*_0 and c^*_0. Here, the prover just provides c^*_0 in the clear, and the verifier checks $c^*_0 G^*_0 = C^*$. Computing c^*_0 required being able to compute a linear combination of $\{c_0 ... c_{n-1}\}$ that was not known ahead of time, so providing it and verifying it convinces the verifier that the prover actually does know all the coefficients that go into the commitment. This concludes the proof.

Recapping:

- The statement we are proving is that C is a commitment to *some* polynomial $P(x) = \sum_i c_i x^i$ committed to using the agreed-upon base points $\{G_0 ... G_{n-1}\}$
- The proof consists of $\log(n)$ pairs of (L, R) values, representing the yellow and green areas at each step. The prover also provides the final c^*_0
- The verifier walks through the proof, generating the α value at each step using the same algorithm as the prover and computing the new C' and G'_i values (the verifier doesn't know the c_i values so they can't compute any c'_i values)
- At the end, they check whether or not $c^*_0 G^*_0 = C^*$

On the whole, the proof contains $2 * \log(n)$ elliptic curve points and one number (for pedants: one *field element*). Verifying the proof takes logarithmic time in every step except one: computing the new G'_i values. This step is, unfortunately, linear.

See also: [Dankrad Feist's more detailed explanation](#) of inner product arguments.

Extension to polynomial evaluations

We can extend to polynomial evaluations with a simple clever trick. Suppose we are trying to prove $P(z) = a$. The prover and the verifier can extend the base points $G_0 ... G_{n-1}$ by attaching powers of z to them: the new base points become $(G_0, 1), (G_1, z) ... (G_{n-1}, z^{n-1})$. These pairs can be treated as mathematical objects (for pedants: *group elements*) much like elliptic curve points themselves; to add them you do so element-by-element: $(A, x) + (B, y) =$ $(A + B,\ x + y)$, using elliptic curve addition for the points and regular field addition for the numbers.

We can make a Pedersen commitment using this extended base!
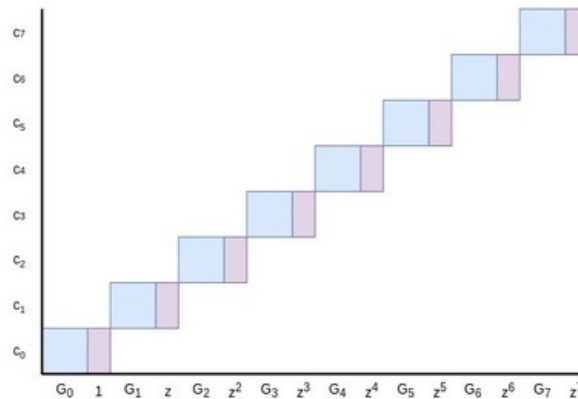
Now, here's a puzzle. Suppose $P(x) = \sum_i c_i x^i$, where $P(z) = a$, would have a commitment $C = \sum_i c_i G_i$ if we were to use the regular elliptic curve points we used before as a base. If we use the pairs (G_i, z^i) as a base instead, the commitment would be (C, y) for some y. What must be the value of y?

The answer is: it must be equal to a! This is easy to see: the commitment is $(C, y) = \sum_i c_i (G_i, z^i)$, which we can decompose as $(\sum_i c_i G_i,\ \sum_i c_i z^i)$. The former is equal to C, and the latter is just the evaluation $P(z)$!

Hence, if C is a "regular" commitment to P using $\{G_0 ... G_{n-1}\}$ as a base, then to prove that $P(z) = a$ we need only use the same protocol above, but proving that (C, a) is a valid commitment using $(G_0, 1), (G_1, z) ... (G_{n-1}, z^{n-1})$ as a base!

Note that in practice, this is usually done slightly differently as an optimization: instead of attaching the numbers to the points and explicitly dealing with structures of the form (G_i, z^i), we add another randomly chosen base point H and express it as $G_i + z^i H$. This saves space.

See [here](#) for an example implementation of this whole protocol.

So, how do we combine these proofs?

Suppose that you are given two polynomial evaluation proofs, with different polynomials and different evaluation points, and want to make a proof that they are both correct. You have:

- Proof Π_1 proving that $P_1(z_1) = y_1$, where P_1 is represented by $com(P_1) = C_1$
- Proof Π_2 proving that $P_2(z_2) = y_2$, where P_2 is represented by $com(P_2) = C_2$

Verifying each proof takes linear time. We want to make a proof that proves that both proofs are correct. This will still take linear time, but the verifier will only have to make *one* round of linear time verification instead of two.

We start off with an observation. The only linear-time step in performing the verification of the proofs is computing the G'_i values. This is $O(n)$ work because you have to combine $\frac{n}{2}$ pairs of G_i values into G'_i values, then $\frac{n}{4}$ pairs of G'_i values into G''_i values, and so on, for a total of n combinations of pairs. But if you look at the algorithm carefully, you will notice that *we don't actually need any of the intermediate G'_i values; we only need the final G^*_0*. This G^*_0 is a linear combination of the initial G_i values. What are the

coefficients to that linear combination? It turns out that the G_i coefficient is the X^i term of this polynomial:

$$[(X + \alpha_1) * (X^2 + \alpha_2)\ *\ ...\ *\ (X^{\frac{n}{2}} + \alpha_{\log(n)})]$$

This is using the $C' = \alpha C + \alpha^2 L + R$ version we mentioned above. The ability to directly compute G^*_0 as a linear combination already cuts down our work to $O(\frac{n}{\log(n)})$ due to [fast linear combination algorithms](), but we can go further.

The above polynomial has degree $n - 1$, with n nonzero coefficients. But its un-expanded form has size $\log(n)$, and so you can *evaluate* the polynomial at any point in $O(\log(n))$ time. Additionally, you might notice that G^*_0 is a commitment to this polynomial, so we can directly *prove* evaluations! So here is what we do:

- The prover computes the above polynomial for each proof; we'll call these polynomials K_1 with $com(K_1) = D_1$ and K_2 with $com(K_2) = D_2$. In a "normal" verification, the verifier would be computing D_1 and D_2 themselves as these are just the G^*_0 values for their respective proofs. Here, the prover provides D_1 and D_2 and the rest of the work is proving that they're correct.
- To prove the correctness of D_1 and D_2 we'll prove that they're correct at a random point. We choose a random point t, and evaluate both $e_1 = K_1(t)$ and $e_2 = K_2(t)$
- The prover generates a random linear combination $L(x) = K_1(x) + rK_2(x)$ (and the verifier can generate $com(L) = D_1 + rD_2$). The prover now just needs to make a single proof that $L(t) = e_1 + re_2$.
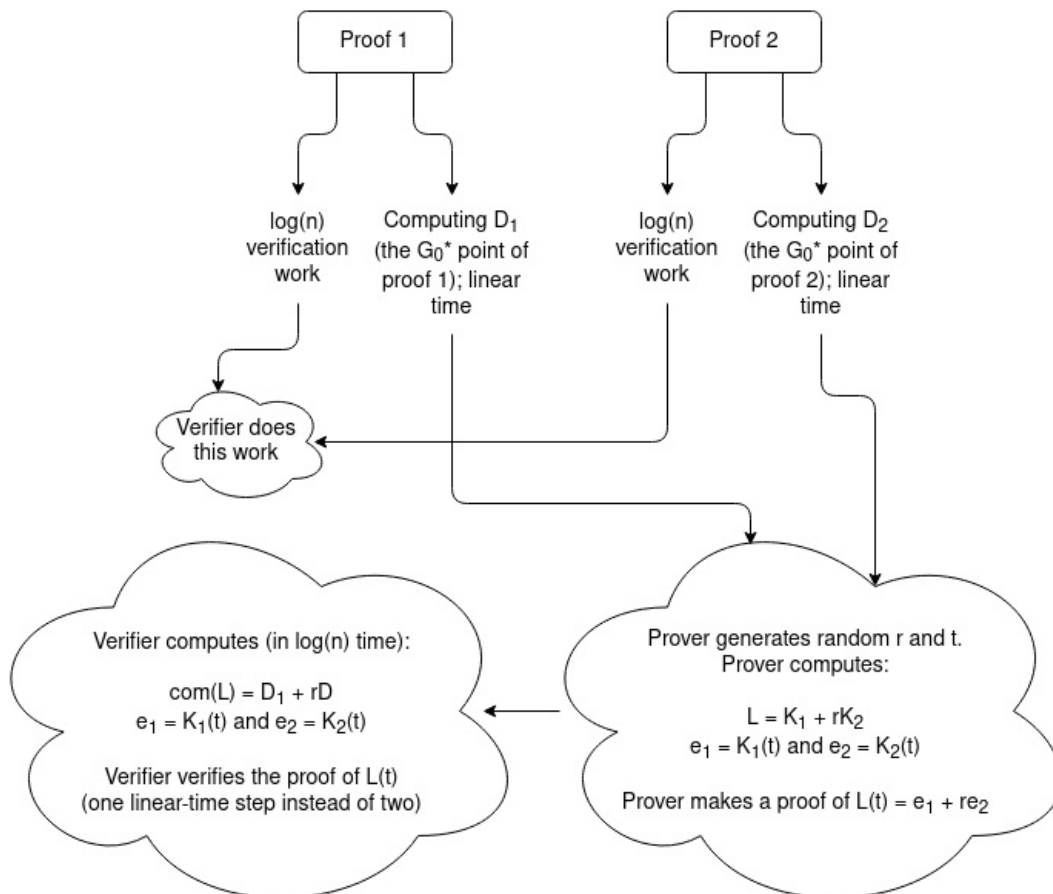
The verifier still needs to do a bunch of extra steps, but all of those steps take either $O(1)$ or $O(\log(n))$ work: evaluate $e_1 = K_1(t)$ and $e_2 = K_2(t)$, calculate the α_i coefficients of both K_i polynomials in the first place, do the elliptic curve addition $com(L) = D_1 + rD_2$. But this all takes vastly less than linear time, so all in all we still benefit: the verifier only needs to do the linear-time step of computing a G^*_0 point themselves once.

This technique can easily be generalized to merge $m > 2$ signatures.

From merging IPAs to merging IPA-based SNARKs: Halo

Now, we get into the core mechanic of the [Halo protocol](#) being integrated in Zcash, which uses this proof combining technique to create a recursive proof system. The setup is simple: suppose you have a chain, where each block has an associated IPA-based SNARK (see [here](#) for how generic SNARKs from polynomial commitments work) proving its correctness. You want to create a new block, building on top of the previous tip of the chain. The new block should have its own IPA-based SNARK proving the correctness of the block. In fact, this proof should cover both the correctness of the new block *and* the correctness of the previous block's proof of the correctness of the entire chain before it.

IPA-based proofs by themselves cannot do this, because a proof of a statement takes longer to verify than checking the statement itself, so a proof of a proof will take even longer to verify than both proofs separately. But proof merging can do it!
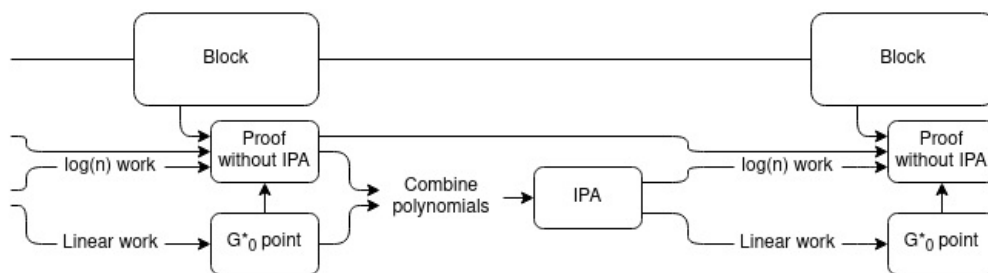
Essentially, we use the usual "recursive SNARK" technique to verify the proofs, except the "proof of a proof" part is only proving the logarithmic part of the work. We add an extra chain of aggregate proofs, using a trick similar to the proof merging scheme above, to handle the linear part of the work. To verify the whole chain, the verifier need only verify one linear-time proof at the very tip of the chain.

The precise details are somewhat different from the exact proof-combining trick in the previous section for efficiency reasons. Instead of using the proof-combining trick to combine multiple proofs, we use it on a *single* proof, just to re-randomize the point that the polynomial committed to by G^*_0 needs to be evaluated at. We then use the *same* newly chosen evaluation point to evaluate the polynomials in the proof of the block's correctness, which allows us to prove the polynomial evaluations together in a single IPA.

Expressed in math:

- Let $P(z) = a$ be the previous statement that needs to be proven
- The prover generates G^*_0
- The prover proves the correctness of the new block plus the logarithmic work in the previous statements by generating a [PLONK proof](#): $Q\_L * A + Q\_R * B + Q\_O * C + Q\_M * A * B + Q\_C = Z * H$
- The prover chooses a random point t, and proves the evaluation of a linear combination of $\{G^*\_0,\ Q\_L,\ A,\ Q\_R,\ B,\ Q\_O,\ C,\ Q\_M,\ Q\_C,\ Z,\ H\}$ at t. We can then check the above equation, replacing each polynomial with its now-verified evaluation at t, to verify the PLONK proof.

Incremental verification, more generally

The size of each "step" does not need to be a full block verification; it could be something as small as a single step of a virtual machine. The smaller the steps the better: it ensures that the linear work that the verifier ultimately has to do at the end is less. The only lower bound is that each step has to be big enough to contain a SNARK verifying the $log(n)$ portion of the work of a step.

But regardless of the fine details, this mechanism allows us to make succinct and easy-to-verify SNARKs, including easy support for recursive proofs that allow you to extend proofs in real time as the computation extends and even have different provers to do different parts of the proving work,

all without pairings or a trusted setup! The main downside is some extra technical complexity, compared with a "simple" polynomial-based proof using eg. KZG-based commitments.

Technology	Cryptographic assumptions	Proof size	Verification time
FRI	Hashes only (quantum safe!)	Large (10-200 kB)	Medium (poly-logarithmic)
Inner product arguments (IPAs)	Basic elliptic curves	Medium (1-3 kB)	**Very high (linear)**
KZG commitments	Elliptic curves + pairings + trusted setup	Short (~500 bytes)	Low (constant)
IPA + Halo-style aggregation	**Basic elliptic curves**	**Medium (1-3 kB)**	**Medium (constant but higher than KZG)**

Not just polynomials! Merging R1CS proofs

A common alternative to building SNARKs out of polynomial games is building SNARKs out of matrix-vector multiplication games. Polynomials and vectors+matrices are both natural bases for SNARK protocols because they are mathematical abstractions that can store and compute over large amounts of data at the same time, and that admit commitment schemes that allow verifiers to check equations quickly.

In R1CS (see a more detailed description [here](#)), an instance of the game consists of three matrices A, B, C, and a solution is a vector Z such that $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$ (the problem is often in practice restricted further by requiring the prover to make part of Z public and requiring the last entry of Z to be 1).

An R1CS instance with a single constraint (so A, B and C have width 1), with a satisfying Z vector, though notice that here the Z appears on the left and has 1 in the top position instead of the bottom.

Just like with polynomial-based SNARKs, this R1CS game can be turned into a proof scheme by creating commitments to A, B and C, requiring the prover to provide a commitment to (the private portion of) Z, and using fancy proving tricks to prove the equation $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$, where $\circ$ is item-by-item multiplication, without fully revealing any of these objects. And just like with IPAs, this R1CS game has a proof merging scheme!

Ioanna Tzialla et al describe such a scheme in a [recent paper](#) (see page 8-9 for their description). They first modify the game by introducing an expanded equation:

$$ (A \cdot Z) \circ (B \cdot Z) - u * (C \cdot Z) = E$$

For a "base" instance, $u = 1$ and $E = 0$, so we get back the original R1CS equation. The extra slack variables are added to make aggregation possible; aggregated instances will have other values of u and E. Now, suppose that you have two solutions to the same instance, though with different u and E variables:

$$(A \cdot Z_1) \circ (B \cdot Z_1) - u_1 * (C \cdot Z_1) = E_1$$

$$(A \cdot Z_2) \circ (B \cdot Z_2) - u_2 * (C \cdot Z_2) = E_2$$

The trick involves taking a random linear combination $Z_3 = Z_1 + r Z_2$, and making the equation work with this new value. First, let's evaluate the left side:

$$ (A \cdot (Z_1 + rZ_2)) \circ (B \cdot (Z_1 + rZ_2)) - (u_1 + ru_2)*(C \cdot (Z_1 + rZ_2)) $$

This expands into the following (grouping the 1, r and r^2 terms together):

$$(A \cdot Z_1) \circ (B \cdot Z_1) - u_1 * (C \cdot Z_1)$$

$$r((A \cdot Z_1) \circ (B \cdot Z_2) + (A \cdot Z_2) \circ (B \cdot Z_1) - u_1 * (C \cdot Z_2) - u_2 * (C \cdot Z_1))$$

$$r^2((A \cdot Z_2) \circ (B \cdot Z_2) - u_2 * (C \cdot Z_2))$$

The first term is just E_1; the third term is $r^2 * E_2$. The middle term is very similar to the cross-term (the yellow + green areas) near the very start of this post. The prover simply provides the middle term (without the r factor), and just like in the IPA proof, the randomization forces the prover to be honest.

Hence, it's possible to make merging schemes for R1CS-based protocols too. Interestingly enough, we don't even technically need to have a "succinct" protocol for proving the $$ (A \cdot Z) \circ (B \cdot Z) = u * (C \cdot Z) + E$$ relation at the end; instead, the prover could just prove by opening all the commitments directly! This would still be "succinct" because the verifier would only need to verify one proof that actually represents an arbitrarily large number of statements. However, in practice having a succinct protocol for this last step is better because it keeps the proofs smaller, and [Tzialla et al's paper](#) provides such a protocol too (see page 10).

Recap

- We don't know of a way to make a commitment to a size-n polynomial where evaluations of the polynomial can be verified in $< O(n)$ time directly. The best that we can do is make a $\log(n)$ sized proof, where all of the work to verify it is logarithmic *except for one final $O(n)$-time piece*.
- But what we *can* do is merge multiple proofs together. Given m proofs of evaluations of size-n polynomials, you can make a proof that covers *all* of these evaluations, that takes logarithmic work plus a single size-n polynomial proof to verify.
- With some clever trickery, separating out the logarithmic parts from the linear parts of proof verification, we can leverage this to make recursive SNARKs.
- These recursive SNARKs are actually more efficient than doing recursive SNARKs "directly"! In fact, even in contexts where direct recursive SNARKs are possible (eg. proofs with KZG commitments), Halo-style techniques are typically used instead because they are more efficient.
- It's not just about polynomials; other games used in SNARKs like R1CS can also be aggregated in similar clever ways.
- No pairings or trusted setups required!

The march toward faster and more efficient and safer ZK-SNARKs just keeps going...

Crypto Cities

Special thanks to Mr Silly and Tina Zhen for early feedback on the post, and to a big long list of people for discussion of the ideas.

One interesting trend of the last year has been the growth of interest in local government, and in the idea of local governments that have wider variance and do more experimentation. Over the past year, Miami mayor Francis Suarez has pursued a [Twitter-heavy](#) tech-startup-like strategy of attracting interest in the city, frequently [engaging](#) with the mainstream [tech industry](#) and [crypto community](#) on Twitter. Wyoming now has a [DAO-friendly legal structure](#), Colorado is [experimenting with quadratic voting](#), and we're seeing more and more experiments making more [pedestrian-friendly street environments](#) for the offline world. We're even seeing projects with varying degrees of radicalness - [Cul de sac](#), [Telosa](#), [CityDAO](#), [Nkwashi](#), [Prospera](#) and many more - trying to create entire neighborhoods and cities from scratch.

Another interesting trend of the last year has been the rapid mainstreaming of crypto ideas such as coins, non-fungible tokens and decentralized autonomous organizations (DAOs). So what would happen if we combine the two trends together? Does it make sense to have a city with a coin, an NFT, a DAO, some record-keeping on-chain for anti-corruption, or even all four? As it turns out, there are already people trying to do just that:

- **[CityCoins.co](#)**, a project that sets up coins intended to become local media of exchange, where a portion of the issuance of the coin goes to the city government. [MiamiCoin](#) already exists, and "San Francisco Coin" appears to be coming soon.
- Other experiments with coin issuance (eg. see [this project in Seoul](#))
- **Experiments with NFTs**, often as a way of funding local artists. [Busan](#) is hosting a government-backed conference exploring what they could do with NFTs.
- **Reno mayor Hillary Schieve's [expansive](#) vision for [blockchainifying](#) the city**, including [NFT sales](#) to support local art, a RenoDAO with RenoCoins issued to local residents that could get revenue from the government renting out properties, blockchain-secured lotteries, blockchain voting and more.
- Much more ambitious projects **creating crypto-oriented cities from scratch**: see [CityDAO](#), which describes itself as, well, "building a city on the Ethereum blockchain" - DAOified governance and all.

But are these projects, in their current form, good ideas? Are there any changes that could make them into *better* ideas? Let us find out...

Why should we care about cities?

Many national governments around the world are showing themselves to be inefficient and slow-moving in response to long-running problems and rapid changes in people's underlying needs. In short, many national governments are missing [live players](#). Even worse, many of the outside-the-box political ideas that *are* being considered or implemented for national governance today are honestly quite terrifying. Do *you* want the USA to be [taken over by a clone of WW2-era Portuguese dictator Antonio Salazar](#), or perhaps an ["American Caesar"](#), to beat down the evil scourge of American leftism? For every idea that can be reasonably described as freedom-expanding or democratic, there are ten that are just different forms of centralized control and walls and universal surveillance.

Now consider local governments. **Cities and states, as we've seen from the examples at the start of this post, are at least in theory capable of genuine dynamism**. There are large and very real differences of culture between cities, so it's easier to find a single city where there is public interest in adopting any particular radical idea than it is to convince an entire country to accept it. There are very real challenges and opportunities in local public goods, urban planning, transportation and many other sectors in the governance of cities that could be addressed. Cities have tightly cohesive internal economies where things like widespread cryptocurrency adoption could realistically independently happen. Furthermore, it's less likely that experiments within cities will lead to terrible outcomes both because cities are regulated by higher-level governments and because cities have an easier escape valve: people who are unhappy with what's going on can more easily exit.

So all in all, it seems like the local level of government is a very undervalued one. And given that [criticism](#) of existing [smart city initiatives](#) often heavily focuses on concerns around centralized governance, lack of transparency and [data privacy](#), blockchain and cryptographic technologies seem like a promising key ingredient for a more open and participatory way forward.

What are city projects up to today?

Quite a lot actually! Each of these experiments is still small scale and largely still trying to find its way around, but they are all at least seeds that could turn into interesting things. Many of the most advanced projects are in the United States, but there is interest across the world; over in Korea the government of Busan is [running an NFT conference](#). Here are a few examples of what is being done today.

Blockchain experiments in Reno

Reno, Nevada mayor [Hillary Schieve](#) is a blockchain fan, focusing primarily on the Tezos ecosystem, and she has recently been exploring blockchain-related ideas (see [her podcast here](#)) in the governance of her city:

- **Selling NFTs to fund local art**, starting with an NFT of [the "Space Whale" sculpture](#) in the middle of the city
- **Creating a Reno DAO**, governed by Reno coins that Reno residents would be eligible to receive via an airdrop. The Reno DAO could start to get sources of revenue; one [proposed idea](#) was the city renting out properties that it owns and the revenue going into a DAO
- **Using blockchains to secure all kinds of processes**: blockchain-secured random number generators for casinos, blockchain-secured [voting](#), etc.


Reno space whale. [Source here](#).

CityCoins.co

CityCoins.co is a project built on Stacks, a blockchain run by an unusual "proof of transfer" (for some reason abbreviated PoX and not PoT) [block production algorithm](#) that is built around the Bitcoin blockchain and ecosystem. 70% of the coin's supply is generated by an ongoing sale mechanism: anyone with STX (the Stacks native token) can send their STX to the city coin contract to generate city coins; the STX revenues are distributed to *existing* city coin holders who stake their coins. The remaining 30% is made available to the city government.

CityCoins has made the interesting decision of trying to make an economic model that does not depend on any government support. The local government does not need to be involved in creating a CityCoins.co coin; a community group can launch a coin by themselves. An [FAQ](#)-provided answer to "What can I do with CityCoins?" includes examples like "CityCoins communities will create apps that use tokens for rewards" and "local businesses can provide discounts or benefits to people who ... stack their CityCoins". In practice, however, the MiamiCoin community is not going at it alone; the Miami government has already [de-facto publicly endorsed it](#).
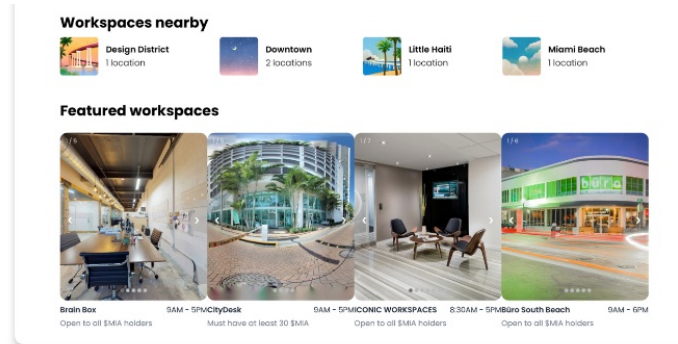
[MiamiCoin hackathon](#) winner: a site that allows coworking spaces to give preferential offers to MiamiCoin holders.

CityDAO

CityDAO is the most radical of the experiments: Unlike Miami and Reno, which are existing cities with existing infrastructure to be upgraded and people to be convinced, CityDAO a DAO with legal status under the Wyoming [DAO law](#) (see their docs [here](#)) trying to create entirely new cities from scratch.

So far, the project is still in its early stages. The team is currently finalizing [a purchase](#) of their first plot of land in a [far-off corner of Wyoming](#). The plan is to start with this plot of land, and then add other plots of land in the future, to build cities, governed by a DAO and making heavy use of [radical economic ideas like Harberger taxes](#) to allocate the land, make collective decisions and manage resources. Their DAO is one of the progressive few that is [avoiding coin voting governance](#); instead, the governance is a voting scheme based on "citizen" NFTs, and ideas have been floated to further limit votes to one-per-person by using [proof-of-humanity verification](#). The NFTs are currently being sold to crowdfund the project; you can buy them on OpenSea.

What do I think cities could be up to?

Obviously there are a lot of things that cities could do in principle. They could add more bike lanes, they could use [CO2 meters](#) and [far-UVC light](#) to more effectively reduce COVID spread without inconveniencing people, and they could even fund life extension research. But my primary specialty is blockchains and this post is about blockchains, so... let's focus on blockchains.

I would argue that there are two distinct categories of blockchain ideas that make sense:

1. Using blockchains to create **more trusted, transparent and verifiable versions of existing processes**.
2. Using blockchains to implement **new and experimental forms of ownership** for land and other scarce assets, as well as **new and experimental forms of democratic governance**.

There's a natural fit between blockchains and both of these categories. Anything happening on a blockchain is very easy to publicly verify, with lots of ready-made freely available tools to help people do that. Any application built on a blockchain can immediately plug in to and interface with other applications in the entire global blockchain ecosystem. Blockchain-based systems are efficient in a way that paper is not, and publicly verifiable in a way that centralized computing systems are not - a necessary combination if you want to, say, make a new form of voting that allows citizens to give high-volume real-time feedback on hundreds or thousands of different issues.

So let's get into the specifics.

What are some existing processes that blockchains could make more trusted and transparent?

One simple idea that plenty of people, including government officials around the world, have brought up to me on many occasions is the idea of governments creating a whitelisted internal-use-only stablecoin for tracking internal government payments. Every tax payment from an individual or organization could be tied to a publicly visible on-chain record minting that number of coins (if we want individual tax payment quantities to be private, there are zero-knowledge ways to make only the total public but still convince everyone that it was computed correctly). Transfers between departments could be done "in the clear", and the coins would be redeemed only by individual contractors or employees claiming their payments and salaries.
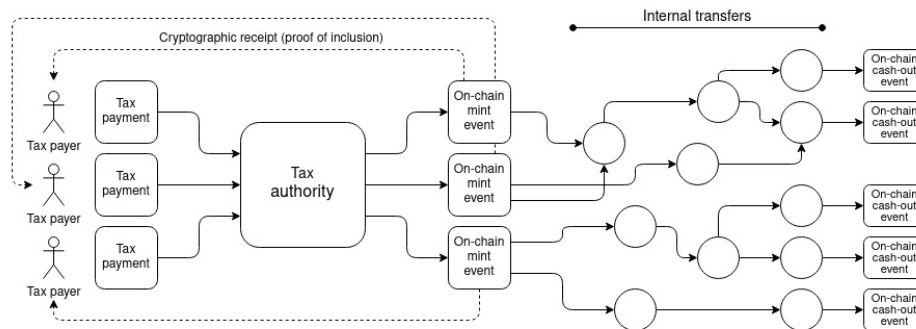
This system could easily be extended. For example, procurement processes for choosing which bidder wins a government contract could largely be done on-chain.

Many more processes could be made more trustworthy with blockchains:

- **Fair random number generators (eg. for lotteries)** - VDFs, such as the one Ethereum is expected to include, could serve as a fair random number generator that could be used to make government-run lotteries more trustworthy. Fair randomness could also be used for many other use cases, such as sortition as a form of government.
- **Certificates**, for example cryptographic proofs that some particular individual is a resident of the city, could be done on-chain for added verifiability and security (eg. if such certificates are issued on-chain, it would become obvious if a large number of false certificates are issued). This can be used by all kinds of local-government-issued certificates.
- **Asset registries**, for land and other assets, as well as more complicated forms of property ownership such as development rights. Due to the need for courts to be able to make assignments in exceptional situations, these registries will likely never be fully decentralized bearer instruments in the same way that cryptocurrencies are, but putting records on-chain can still make it easier to see what happened in what order in a dispute.

Eventually, even **voting** could be done on-chain. Here, many complexities and dragons loom and it's really important to be careful; a sophisticated solution combining blockchains, zero knowledge proofs and other cryptography is needed to achieve all the desired privacy and security properties. However, if humanity is ever going to move to electronic voting at all, local government seems like the perfect place to start.

What are some radical economic and governance experiments that could be interesting?

But in addition to these kinds of blockchain overlays onto things that governments *already* do, we can also look at blockchains as an opportunity for governments to make completely *new* and radical experiments in economics and governance. These are not necessarily final ideas on what I think should be done; they are more initial explorations and suggestions for possible directions. Once an experiment starts, real-world feedback is often by far the most useful variable to determine how the experiment should be adjusted in the future.

Experiment #1: a more comprehensive vision of city tokens

CityCoins.co is one vision for how city tokens could work. But it is far from the only vision. Indeed, the CityCoins.so approach has significant risks, particularly in how economic model is heavily tilted toward early adopters. 70% of the STX revenue from minting new coins is given to *existing stakers of the city coin*. More coins will be issued in the next five years than in the fifty years that follow. It's a good deal for the government in 2021, but what about 2051? Once a government endorses a particular city coin, it becomes difficult for it to change directions in the future. Hence, it's important for city governments to think carefully about these issues, and choose a path that makes sense for the long term.

Here is a different possible sketch of a narrative of how city tokens might work. It's far from the *only* possible alternative to the CityCoins.co vision; see Steve Waldman's excellent article arguing for a city-localized medium of exchange for yet another possible direction. In any case, city tokens are a wide design space, and there are many different options worth considering. Anyway, here goes...

The concept of home ownership in its current form is a notable double-edged sword, and the specific ways in which it's actively encouraged and legally structured is considered by many to be one of the biggest economic policy mistakes that we are making today. **There is an inevitable political tension between a home as a place to live and a home as an investment asset**, and the pressure to satisfy communities who care about the latter often ends up severely harming the affordability of the former. A resident in a city either owns a home, making them massively over-exposed to land prices and introducing perverse incentives to fight against construction of new homes, or they rent a home, making them *negatively* exposed to the real estate market and thus putting them economically at odds with the goal of making a city a nice place to live.

But even despite all of these problems, many still find home ownership to be not just a good personal choice, but something worthy of actively subsidizing or socially encouraging. One big reason is that it nudges people to save money and build up their net worth. Another big reason is that despite its flaws, it creates economic alignment between residents and the communities they live in. **But what if we could give people a way to save and create that economic alignment without the flaws?** What if we could create a divisible and fungible city token, that residents could hold as many units of as they can afford or feel comfortable with, and whose value goes up as the city prospers?

First, let's start with some possible objectives. Not all are necessary; a token that accomplishes only three of the five is already a big step forward. But we'll try to hit as many of them as possible:

- **Get sustainable sources of revenue for the government**. The city token economic model should avoid redirecting *existing* tax revenue; instead, it should find *new* sources of revenue.
- **Create economic alignment between residents and the city**. This means first of all that the coin itself should clearly become more valuable as the city becomes more attractive. But it also means that the economics should actively encourage *residents* to hold the coin more than faraway hedge funds.
- **Promote saving and wealth-building**. Home ownership does this: as home owners make mortgage payments, they build up their net worth by default. City tokens could do this too, making it attractive to accumulate coins over time, and even gamifying the experience.
- **Encourage more pro-social activity**, such as positive actions that help the city and more sustainable use of resources.
- **Be egalitarian**. Don't unduly favor wealthy people over poor people (as badly designed economic mechanisms often do accidentally). A token's

divisibility, avoiding a sharp binary divide between haves and have-nots, does a lot already, but we can go further, eg. by allocating a large portion of new issuance to residents as a UBI.

One pattern that seems to easily meet the first three objectives is providing benefits to holders: if you hold at least X coins (where X can go up over time), you get some set of services for free. MiamiCoin is trying to encourage businesses to do this, but we could go further and make *government* services work this way too. One simple example would be making existing public parking spaces only available for free to those who hold at least some number of coins in a locked-up form. This would serve a few goals at the same time:

- Create an **incentive to hold the coin**, sustaining its value.
- Create an **incentive specifically for *residents* to hold the coin**, as opposed to otherwise-unaligned faraway investors. Furthermore, the incentive's usefulness is capped per-person, so it encourages widely distributed holdings.
- Creates **economic alignment** (city becomes more attractive -> more people want to park -> coins have more value). **Unlike home ownership, this creates alignment with an *entire town***, and not merely a very specific location in a town.
- **Encourage sustainable use of resources**: it would reduce usage of parking spots (though people without coins who really need them could still pay), supporting many local governments' desires to open up more space on the roads to be more pedestrian-friendly. Alternatively, restaurants could also be allowed to lock up coins through the same mechanism and claim parking spaces to use for outdoor seating.

But to avoid perverse incentives, it's extremely important to avoid overly depending on one specific idea and instead to have a diverse array of possible revenue sources. **One excellent gold mine of places to give city tokens value, and at the same time experiment with novel governance ideas, is zoning**. If you hold at least Y coins, then you can quadratically vote on the fee that nearby landowners have to pay to bypass zoning restrictions. This hybrid market + direct democracy based approach would be much more efficient than current overly cumbersome permitting processes, and the fee itself would be another source of government revenue. More generally, any of the ideas in the next section could be combined with city tokens to give city token holders more places to use them.

Experiment #2: more radical and participatory forms of governance

This is where Radical Markets ideas such as Harberger taxes, quadratic voting and quadratic funding come in. I already brought up some of these ideas in the section above, but you don't have to have a dedicated city token to do them. Some limited government use of quadratic voting and funding has already happened: see the Colorado Democratic party and the Taiwanese presidential hackathon, as well as not-yet-government-backed experiments like Gitcoin's Boulder Downtown Stimulus. But we could do more!

One obvious place where these ideas can have long-term value is giving developers incentives to improve the **aesthetics of buildings** that they are building (see here, here, here and here for some recent examples of professional blabbers debating the aesthetics of modern architecture). Harberger taxes and other mechanisms could be used to radically reform **zoning** rules, and blockchains could be used to administer such mechanisms in a more trustworthy and efficient way. Another idea that is more viable in the short term is **subsidizing local businesses**, similar to the Downtown Stimulus but on a larger and more permanent scale. Businesses produce various kinds of positive externalities in their local communities all the time, and those externalities could be more effectively rewarded. **Local news** could be quadratically funded, revitalizing a long-struggling industry. Pricing for **advertisements** could be set based on real-time votes of how much people enjoy looking at each particular ad, encouraging more originality and creativity.

More democratic feedback (and possibly even retroactive democratic feedback!) could plausibly create better incentives in all of these areas. And **21st-century digital democracy through real-time online quadratic voting and funding could plausibly do a much better job than 20th-century democracy, which seems in practice to have been largely characterized by rigid building codes and obstruction at planning and permitting hearings**. And of course, if you're going to use blockchains to secure voting, starting off by doing it with fancy new kinds of votes seems far more safe and politically feasible than re-fitting existing voting systems.

Mandatory solarpunk picture intended to evoke a positive image of what might happen to our cities if real-time quadratic votes could set subsidies and prices for everything.

Conclusions

There are a lot of worthwhile ideas for cities to experiment with that could be attempted by existing cities or by new cities. New cities of course have the advantage of not having existing residents with existing expectations of how things should be done; but the concept of creating a new city itself is, in modern times, relatively untested. Perhaps the multi-billion-dollar capital pools in the hands of people and projects enthusiastic to try new things could get us over the hump. But even then, existing cities will likely continue to be the place where most people live for the foreseeable future, and existing cities can use these ideas too.

Blockchains can be very useful in both the more incremental and more radical ideas that were proposed here, even despite the inherently "trusted" nature of a city government. Running any new or existing mechanism on-chain gives the public an easy ability to verify that everything is following the rules. Public chains are better: the benefits from existing infrastructure for users to independently verify what is going on far outweigh the losses from transaction fees, which are expected to quickly decrease very soon from rollups and sharding. If strong privacy is required, blockchains can be combined zero knowledge cryptography to give privacy and security at the same time.

The main trap that governments should avoid is too quickly sacrificing optionality. An *existing* city could fall into this trap by launching a bad city token instead of taking things more slowly and launching a good one. A *new* city could fall into this trap by selling off too much land, sacrificing the entire upside to a small group of early adopters. Starting with self-contained experiments, and taking things slowly on moves that are truly irreversible, is ideal. But at the same time, it's also important to seize the opportunity in the first place. There's a lot that can and should be improved with cities, and a lot of opportunities; despite the challenges, crypto cities broadly are an idea whose time has come.

On Nathan Schneider on the limits of cryptoeconomics

2021 Sep 26

Nathan Schneider has recently released [an article](#) describing his perspectives on cryptoeconomics, and particularly on the limits of cryptoeconomic approaches to governance and what cryptoeconomics could be augmented with to improve its usefulness. This is, of course, a topic that is dear to me ([1] [2] [3] [4] [5]), so it is heartening to see someone else take the blockchain space seriously as an intellectual tradition and engage with the issues from a different and unique perspective.

The main question that Nathan's piece is trying to explore is simple. There is a large body of intellectual work that criticizes a bubble of concepts that they refer to as "economization", "neoliberalism" and similar terms, arguing that they corrode democratic political values and leave many people's needs unmet as a result. The world of cryptocurrency is very economic (lots of tokens flying around everywhere, with lots of functions being assigned to those tokens), very neo (the space is 12 years old!) and very liberal (freedom and voluntary participation are core to the whole thing). Do these critiques also apply to blockchain systems? If so, what conclusions should we draw, and how could blockchain systems be designed to account for these critiques? Nathan's answer: more hybrid approaches combining ideas from both economics and politics. But what will it actually take to achieve that, and will it give the results that we want? My answer: yes, but there's a lot of subtleties involved.

What are the critiques of neoliberalism and economic logic?

Near the beginning of Nathan's piece, he describes the critiques of overuse of economic logic briefly. That said, he does not go much further into the underlying critiques himself, preferring to point to other sources that have already covered the issue in depth:

> The economics in cryptoeconomics raises a particular set of anxieties. Critics have long warned against the expansion of economic logics, crowding out space for vigorous politics in public life. From the Zapatista insurgents of southern Mexico (Hayden, 2002) to political theorists like William Davies (2014) and Wendy Brown (2015), the "neoliberal" aspiration for economics to guide all aspects of society represents a threat to democratic governance and human personhood itself. Here is Brown:
>
> > Neoliberalism transmogrifies every human domain and endeavor, along with humans themselves, according to a specific image of the economic. All conduct is economic conduct; all spheres of existence are framed and measured by economic terms and metrics, even when those spheres are not directly monetized. In neoliberal reason and in domains governed by it, we are only and everywhere homo oeconomicus (p. 10)
>
> For Brown and other critics of neoliberalism, the ascent of the economic means the decline of the political—the space for collective determinations of the common good and the means of getting there.

At this point, it's worth pointing out that the "neoliberalism" being criticized here is not the same as the "neoliberalism" that is cheerfully promoted by the [lovely folks at The Neoliberal Project](#); the thing being critiqued here is a kind of "enough two-party trade can solve everything" mentality, whereas The Neoliberal Project favors a mix of markets and democracy. But what is the thrust of the critique that Nathan is pointing to? What's the problem with everyone acting much more like homo oeconomicus? For this, we can take a detour and peek into the source, Wendy Brown's *Undoing the Demos*, itself. The book helpfully provides a list of the top "four deleterious effects" (the below are reformatted and abridged but direct quotes):

- **Intensified inequality**, in which the very top strata acquires and retains ever more wealth, the very bottom is literally turned out on the streets or into the growing urban and sub-urban slums of the world, while the middle strata works more hours for less pay, fewer benefits, less security...

- **Crass or unethical commercialization** of things and activities considered inappropriate for marketization. The claim is that marketization contributes to human exploitation or degradation, [...] limits or stratifies access to what ought to be broadly accessible and shared, [...] or because it enables something intrinsically horrific or severely denigrating to the planet.
- **Ever-growing intimacy of corporate and finance capital with the state**, and corporate domination of political decisions and economic policy
- **Economic havoc wreaked on the economy by the ascendance and liberty of finance capital**, especially the destabilizing effects of the inherent bubbles and other dramatic fluctuations of financial markets.

The bulk of Nathan's article follows along with analyses of how these issues affect DAOs and governance mechanisms within the crypto space specifically. Nathan focuses on three key problems:

- **Plutocracy**: "Those with more tokens than others hold more [I would add, *disproportionately* more] decision-making power than others..."
- **Limited exposure to diverse motivations**: "Cryptoeconomics sees only a certain slice of the people involved. Concepts such as self- sacrifice, duty, and honor are bedrock features of most political and business organizations, but difficult to simulate or approximate with cryptoeconomic incentive design"
- **Positive and negative externalities**: "Environmental costs are classic externalities—invisible to the feedback loops that the system understands and that communicate to its users as incentives ... the challenge of funding"public goods" is another example of an externality - and one that threatens the sustainability of crypteconomic systems"

The natural questions that arise for me are (i) to what extent do I agree with this critique at all and how it fits in with my own thinking, and (ii) how does this affect blockchains, and what do blockchain protocols need to actually do to avoid these traps?

What do I think of the critiques of neoliberalism generally?

I disagree with some, agree with others. I have always been suspicious of criticism of "crass and unethical commercialization", because it frequently feels like the author is attempting to launder their own feelings of disgust and aesthetic preferences into grand ethical and political ideologies - a sin common among all such ideologies, often the right ([random example here](#)) even more than the left. Back in the days when I had much less money and would sometimes walk a full hour to the airport to avoid a taxi fare, I remember thinking that I would love to get compensated for donating blood or using my body for clinical trials. And so to me, the idea that such transactions are inhuman exploitation has never been appealing.

But at the same time, I am far from a [Walter Block-style defender](#) of *all* locally-voluntary two-party commerce. I've written up my own viewpoints expressing similar concerns to parts of Wendy Brown's list in various articles:

- [Multiple](#) pieces [decrying](#) the evils of vote buying, or even [financialized governance generally](#)
- The [importance](#) of [public goods funding](#).
- Failure modes in financial markets due to [subtle issues like capital efficiency](#).

So where does *my own* opposition to mixing finance and governance come from? This is a complicated topic, and my conclusions are in large part a result of my own failure after years of attempts to *find* a financialized governance mechanism that is economically stable. So here goes...

Finance is the absence of collusion prevention

Out of the standard assumptions in what gets pejoratively called "spherical cow economics", people normally tend to focus on the unrealistic nature of *perfect information* and *perfect rationality*. But the unrealistic assumption that is hidden in the list that strikes me as even more misleading is *individual choice*: the idea that each agent is separately making their own decisions, no agent has a positive or negative stake in another agent's outcomes, and there are no "side games"; the only thing that sees each agent's decisions is the black box that we call "the mechanism".
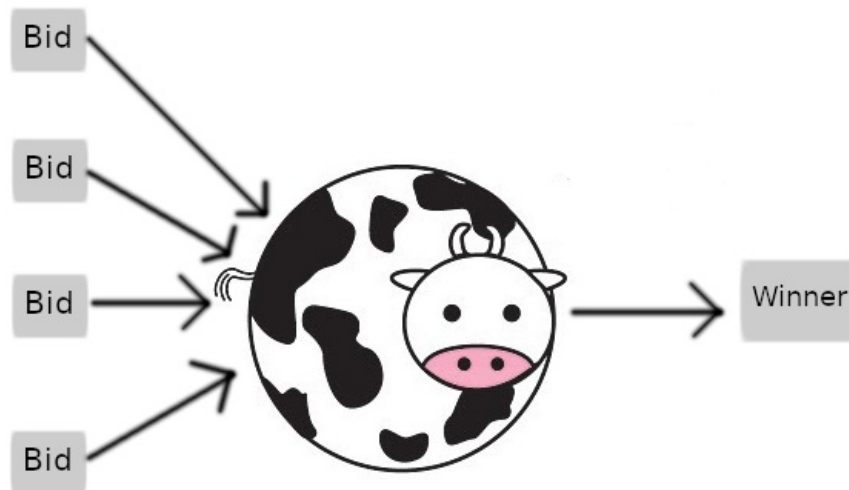
This assumption is often used to bootstrap complex contraptions such as the VCG mechanism, whose theoretical optimality is based on beautiful arguments that because the price each player pays only depends on *other players' bids*, each player has no incentive to make a bid that does not reflect their true value in order to manipulate the price. A beautiful argument in theory, but it breaks down completely once you introduce the possibility that even two of the players are either allies or adversaries outside the mechanism.

Economics, and economics-inspired philosophy, is great at describing the complexities that arise when the number of players "playing the game" increases from one to two (see the tale of Crusoe and Friday in Murray Rothbard's The Ethics of Liberty for one example). But what this philosophical tradition completely misses is that going up to *three* players adds an even further layer of complexity. In an interaction between two people, the two can ignore each other, fight or trade. In an interaction between three people, there exists a new strategy: any two of the three can communicate and band together to gang up on the third. Three is the smallest denominator where it's possible to talk about a 51%+ attack that has someone outside the clique to be a victim.

When there's only two people, more coordination can only be good. But once there's three people, the wrong kind of coordination can be harmful, and techniques to *prevent* harmful coordination (including decentralization itself) can become very valuable. And it's this management of coordination that is the essence of "politics".
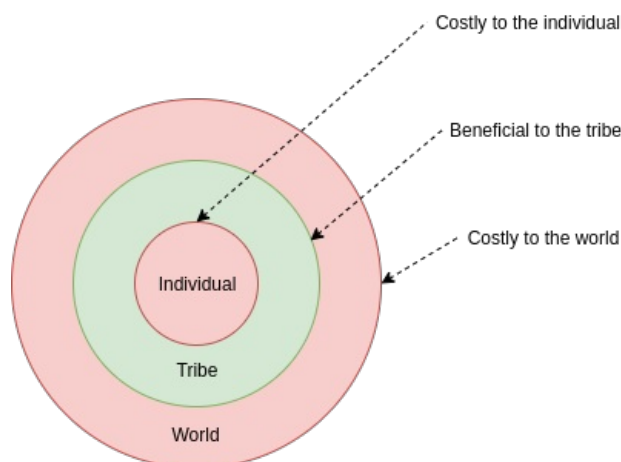
Going from two people to three introduces the possibility of harms from unbalanced coordination: it's not just "the individual versus the group", it's "the individual versus the group versus the world".

Now, we can understand try to use this framework to understand the pitfalls of "finance". **Finance can be viewed as a set of patterns that naturally emerge in many kinds of systems that do not attempt to prevent collusion.** Any system which *claims* to be non-finance, but does not actually make an effort to prevent collusion, will eventually acquire the characteristics of finance, if not something worse. To see why this is the case, compare two point systems we are all familiar with: money, and Twitter likes. Both kinds of points are valuable for extrinsic reasons, both have inevitably become status symbols, and both are number games where people spend a lot of time optimizing to try to get a higher score. And yet, they behave very differently. So what's the fundamental difference between the two?

The answer is simple: it's the lack of an efficient market to enable agreements like "I like your tweet if you like mine", or "I like your tweet if you pay me in some other currency". If such a market existed and was easy to use, Twitter would collapse completely (something like hyperinflation would happen, with the likely outcome that everyone would run automated bots that like every tweet to claim rewards), and even the likes-for-money markets that exist illicitly *today* are a big problem for Twitter. With money, however, "I send X to you if you send Y to me" is not an *attack vector*, it's just a boring old currency exchange transaction. A Twitter clone that does not prevent like-for-like markets would "hyperinflate" into everyone liking everything, and if that Twitter clone tried to stop the hyperinflation by limiting the number of likes each user can make, the likes would behave like a currency, and the end result would behave the same as if Twitter just added a tipping feature.

So what's the problem with finance? Well, if finance is optimized and structured collusion, then we can look for places where finance causes problems by using our existing economic tools to understand which mechanisms break if you introduce collusion! Unfortunately, governance by voting is a central example of this category; I've covered why in the ["moving beyond coin voting governance" post](#) and [many](#) other [occasions](#). Even worse, cooperative game theory suggests that there might be [no possible way to make a fully collusion-resistant governance mechanism](#).

And so we get the fundamental conundrum: the cypherpunk spirit is fundamentally about making maximally immutable systems that work with as little information as possible about who is participating ("on the internet, nobody knows you're a dog"), but making new forms of *governance* requires the system to have richer information about its participants and ability to dynamically respond to attacks in order to remain stable in the face of actors with unforeseen incentives. Failure to do this means that everything looks like finance, which means, well.... perennial over-representation of concentrated interests, and all the problems that come as a result.

On the internet, nobody knows if you're 0.0244 of a dog ([image source](#)). But what does this mean for governance?

The central role of collusion in understanding the difference between Kleros and regular courts

Now, let us get back to Nathan's article. The distinction between financial and non-financial mechanisms is key in the article. Let us start off with a description of the Kleros court:

> The jurors stood to earn rewards by correctly choosing the answer that they expected other jurors to independently select. This process implements the "Schelling point" concept in game theory (Aouidef et al., 2021; Dylag & Smith, 2021). Such a jury does not deliberate, does not seek a common good together; its members unite through self-interest. Before coming to the jury, the factual basis of the case was supposed to come not from official organs or respected news organizations but from anonymous users similarly disciplined by reward-seeking. The prediction market itself was premised on the supposition that people make better forecasts when they stand to gain or lose the equivalent of money in the process. The politics of the presidential election in question, here, had been thoroughly transmuted into a cluster of economies.

The implicit critique is clear: the Kleros court is ultimately motivated to make decisions not on the basis of their "true" correctness or incorrectness, but rather on the basis of their financial interests. If Kleros is deciding whether Biden or Trump won the 2020 election, and one Kleros juror really likes Trump, precommits to voting in his favor, and bribes other jurors to vote the same way, other jurors are likely to fall in line because of Kleros's conformity incentives: jurors are rewarded if their vote agrees with the majority vote, and penalized otherwise. The theoretical answer to this is the right to exit: if the majority of Kleros jurors vote to proclaim that Trump won the election, a minority can spin off a fork of Kleros where Biden is considered to have won, and their fork may well get a higher market price than the original. Sometimes, [this actually works](#)! But, as Nathan points out, it is not always so simple:

> But exit may not be as easy as it appears, whether it be from a social-media network or a protocol. The persistent dominance of early-to-market blockchains like Bitcoin and Ethereum suggests that cryptoeconomics similarly favors incumbency.

But alongside the implicit critique is an implicit promise: that *regular courts* are somehow able to rise above self-interest and "seek a common good together" and thereby avoid some of these failure modes. What is it that financialized Kleros courts lack, but non-financialized regular courts retain, that makes them more robust? One possible answer is that courts lack Kleros's explicit conformity incentive. But if you just take Kleros as-is, remove the conformity incentive (say, there's a reward for voting that does not depend on how you vote), and do nothing else, you risk creating even more problems. Kleros judges could get lazy, but more importantly if there's no incentive at all to choose how you vote, even the tiniest bribe could affect a judge's decision.

So now we get to the real answer: the key difference between financialized Kleros courts and non-financialized regular courts is that financialized Kleros courts are, well... *financialized*. They make no effort to explicitly prevent collusion. Non-financialized courts, on the other hand, do prevent collusion in two key ways:

- Bribing a judge to vote in a particular way is explicitly illegal
- The judge position itself is non-fungible. It gets awarded to specific carefully-selected individuals, and they cannot simply go and sell or reallocate their entire judging rights and salary to someone else.

The only reason why political and legal systems work is that a lot of hard thinking and work has gone on behind the scenes to *insulate the decision-makers from extrinsic incentives*, and punish them explicitly if they are discovered to be accepting incentives from the outside. **The lack of extrinsic motivation allows the intrinsic motivation to shine through. Furthermore, the lack of transferability allows governance power to be given to specific actors whose intrinsic motivations we trust, avoiding governance power always flowing to "the highest bidder".** But in the case of Kleros, the lack of hostile extrinsic motivation cannot be guaranteed, and transferability is unavoidable, and so overpoweringly strong in-mechanism extrinsic motivation (the conformity incentive) was the best solution they could find to deal with the problem.

And of course, the "final backstop" that Kleros relies on, the right of users to fork away, itself depends on social coordination to take place - a messy and difficult institution, often derided by cryptoeconomic purists as "proof of social media", that works precisely because public discussion has lots of informal collusion detection and prevention all over the place.

Collusion in understanding DAO governance issues

But what happens when there is no single right answer that they can expect voters to converge on? This is where we move away from *adjudication* and toward *governance* (yes, I know that adjudication

has unavoidably grey edge cases too. Governance just has them much more often). Nathan writes:

> Governance by economics is nothing new. Joint-stock companies conventionally operate on plutocratic governance—more shares equals more votes. This arrangement is economically efficient for aligning shareholder interests (Davidson and Potts, this issue), even while it may sideline such externalities as fair wages and environmental impacts...

In my opinion, this actually concedes too much! Governance by economics is not "efficient" once you drop the spherical-cow assumption of no collusion, because it is inherently vulnerable to 51% of the stakeholders colluding to liquidate the company and split its resources among themselves. The only reason why this does not happen much more often "in real life" is because of many decades of shareholder regulation that have been explicitly built up to ban the most common types of abuses. This regulation is, of course, non-"economic" (or, in my lingo, it makes corporate governance less *financialized*), because it's an explicit attempt to prevent collusion.

Notably, Nathan's favored solutions do *not* try to regulate coin voting. Instead, they try to limit the harms of its weaknesses by combining it with additional mechanisms:

> Rather than relying on direct token voting, as other protocols have done, The Graph uses a board-like mediating layer, the Graph Council, on which the protocol's major stakeholder groups have representatives. In this case, the proposal had the potential to favor one group of stakeholders over others, and passing a decision through the Council requires multiple stakeholder groups to agree. At the same time, the Snapshot vote put pressure on the Council to implement the will of token-holders.

In the case of 1Hive, the anti-financialization protections are described as being purely cultural:

> According to a slogan that appears repeatedly in 1Hive discussions, "Come for the honey, stay for the bees." That is, although economics figures prominently as one first encounters and explores 1Hive, participants understand the community's primary value as interpersonal, social, and non-economic.

I am personally skeptical of the latter approach: it can work well in low-economic-value communities that are fun oriented, but if such an approach is attempted in a more serious system with widely open participation and enough at stake to invite determined attack, it will not survive for long. As I wrote above, "any system which *claims* to be non-finance, but does not actually make an effort to prevent collusion, will eventually acquire the characteristics of finance".

[Edit/correction 2021.09.27: it has been brought to my attention that in addition to culture, financialization is limited by (i) conviction voting, and (ii) juries enforcing a covenant. I'm skeptical of conviction voting in the long run; many DAOs use it today, but in the long term it can be defeated by wrapper tokens. The covenant, on the other hand, is interesting. My fault for not checking in more detail.]
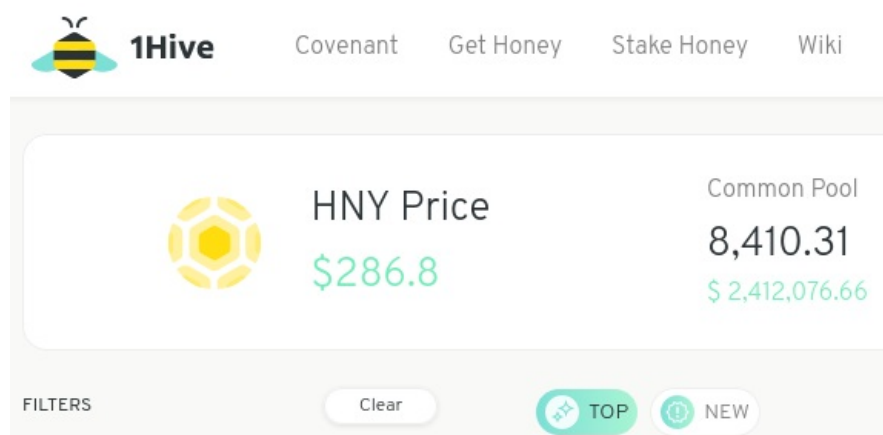
*The money is **called** honey. But is calling money honey enough to make it work differently than money? If not, how much more do you have to do?*

The solution in TheGraph is very much an instance of collusion prevention: the participants have

[been hand-picked](#) to come from diverse constituencies and to be trusted and upstanding people who are unlikely to sell their voting rights. Hence, I am bullish on that approach if it successfully avoids centralization.

So how can we solve these problems more generally?

Nathan's post argues:

> A napkin sketch of classical, never-quite-achieved liberal democracy (Brown, 2015) would depict a market (governed through economic incentives) enclosed in politics (governed through deliberation on the common good). Economics has its place, but the system is not economics all the way down; the rules that guide the market, and that enable it in the first place, are decided democratically, on the basis of citizens' civil rights rather than their economic power. By designing democracy into the base-layer of the system, it is possible to overcome the kinds of limitations that cryptoeconomics is vulnerable to, such as by counteracting plutocracy with mass participation and making visible the externalities that markets might otherwise fail to see.

There is one key difference between blockchain political theory and traditional nation-state political theory - and one where, in the long run, nation states may well have to learn from blockchains. Nation-state political theory talks about "markets embedded in democracy" as though democracy is an encompassing base layer that encompasses all of society. In reality, this is not true: there are multiple countries, and every country at least to some degree permits trade with outside countries whose behavior they cannot regulate. Individuals and companies have choices about which countries they live in and do business in. Hence, markets are not just embedded in democracy, they also surround it, and the real world is a complicated interplay between the two.

Blockchain systems, instead of trying to fight this interconnectedness, embrace it. A blockchain system has no ability to regular "the market" in the sense of people's general ability to freely make transactions. But what it *can* do is regulate and structure (or even create) specific markets, setting up patterns of specific behaviors whose incentives are ultimately set and guided by institutions that have anti-collusion guardrails built in, and can resist pressure from economic actors. And indeed, this is the direction Nathan ends up going in as well. He talks positively about the design of Civil as an example of precisely this spirit:

> The aborted Ethereum-based project Civil sought to leverage cryptoeconomics to protect journalism against censorship and degraded professional standards (Schneider, 2020). Part of the system was the Civil Council, a board of prominent journalists who served as a kind of supreme court for adjudicating the practices of the network's newsrooms. Token holders could earn rewards by successfully challenging a newsroom's practices; the success or failure of a challenge ultimately depended on the judgment of the Civil Council, designed to be free of economic incentives clouding its deliberations. In this way, a cryptoeconomic enforcement market served a non-economic social mission. This kind of design could enable cryptoeconomic networks to serve purposes not reducible to economic feedback loops.

This is fundamentally very similar to an idea that I proposed in 2018: [prediction markets to scale up content moderation](#). Instead of doing content moderation by running a low-quality AI algorithm on all content, with lots of false positives, there could be an open mini prediction market on each post, and if the volume got high enough a high-quality committee could step in an adjudicate, and the prediction market participants would be penalized or rewarded based on whether or not they had correctly predicted the outcome. In the mean time, posts with prediction market scores predicting that the post would be removed would not be shown to users who did not explicitly opt-in to participate in the prediction game. There is precedent for this kind of open but accountable moderation: [Slashdot meta moderation](#) is arguably a limited version of it. This more financialized version of meta-moderation through prediction markets could produce superior outcomes because the incentives invite highly competent and professional participants to take part.

Nathan then expands:

> I have argued that pairing cryptoeconomics with political systems can help overcome the limitations that bedevil cryptoeconomic governance alone. Introducing purpose-centric mechanisms and temporal modulation can compensate for the blind-spots of token economies. But I am not arguing against cryptoeconomics altogether. Nor am I arguing that these sorts of politics must occur in every app and protocol. Liberal democratic theory permits diverse forms of association and business within a democratic structure, and similarly politics may be necessary only at key leverage points in an ecosystem to overcome the limitations of cryptoeconomics alone.

This seems broadly correct. Financialization, as Nathan points out in his conclusion, has benefits in that it attracts a large amount of motivation and energy into building and participating in systems that would not otherwise exist. Furthermore, *preventing* financialization is very difficult and high cost, and works best when done sparingly, where it is needed most. However, it is also true that financialized systems are much more stable if their incentives are anchored around a system that is ultimately non-financial.

Prediction markets avoid the plutocracy issues inherent in coin voting because they introduce *individual accountability*: users who acted in favor of what ultimately turns out to be a bad decision suffer more than users who acted against it. However, a prediction market requires some statistic that it is measuring, and measurement oracles cannot be made secure through cryptoeconomics alone: at the very least, community forking as a backstop against attacks is required. And if we want to avoid the messiness of frequent forks, some other explicit non-financialized mechanism at the center is a valuable alternative.
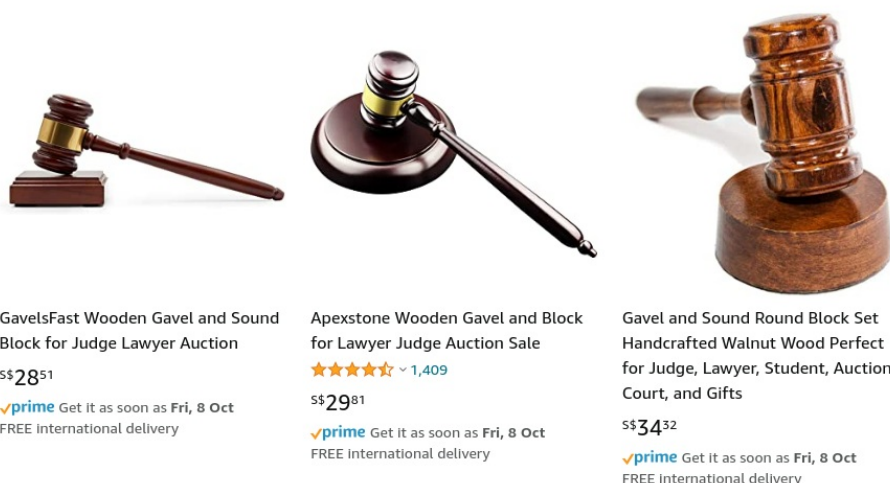
Conclusions

In his conclusion, Nathan writes:

> But the autonomy of cryptoeconomic systems from external regulation could make them even more vulnerable to runaway feedback loops, in which narrow incentives overpower the common good. The designers of these systems have shown an admirable capacity to devise cryptoeconomic mechanisms of many kinds. But for cryptoeconomics to achieve the institutional scope its advocates hope for, it needs to make space for less-economic forms of governance.

> If cryptoeconomics needs a political layer, and is no longer self-sufficient, what good is cryptoeconomics? One answer might be that cryptoeconomics can be the basis for securing more democratic and values-centered governance, where incentives can reduce reliance on military or police power. Through mature designs that integrate with less-economic purposes, cryptoeconomics might transcend its initial limitations. Politics needs cryptoeconomics, too ... by integrating cryptoeconomics with democracy, both legacies seem poised to benefit.

I broadly agree with both conclusions. The language of collusion prevention can be helpful for understanding *why* cryptoeconomic purism so severely constricts the design space. "Finance" is a category of patterns that emerge when systems do not attempt to prevent collusion. When a system does not prevent collusion, it cannot treat different individuals differently, or even different *numbers* of individuals differently: whenever a "position" to exert influence exists, the owner of that position can just sell it to the highest bidder.

GavelsFast Wooden Gavel and Sound Block for Judge Lawyer Auction

S$28⁵¹

prime Get it as soon as **Fri, 8 Oct**
FREE international delivery

Apexstone Wooden Gavel and Block for Lawyer Judge Auction Sale

★★★★⯪ ˅ 1,409

S$29⁸¹

prime Get it as soon as **Fri, 8 Oct**
FREE international delivery

Gavel and Sound Round Block Set Handcrafted Walnut Wood Perfect for Judge, Lawyer, Student, Auction Court, and Gifts

S$34³²

prime Get it as soon as **Fri, 8 Oct**
FREE international delivery

Gavels on Amazon. A world where these were NFTs that actually came with associated judging power may well be a fun one, but I would certainly not want to be a defendant!

The language of [defense-focused design](), on the other hand, is an underrated way to think about

where some of the *advantages* of blockchain-based designs can be. Nation state systems often deal with threats with one of two totalizing mentalities: *closed borders* vs *conquer the world*. A closed borders approach attempts to make hard distinctions between an "inside" that the system *can* regulate and an "outside" that the system cannot, severely restricting flow between the inside and the outside. Conquer-the-world approaches attempt to extraterritorialize a nation state's preferences, seeking a state of affairs where there is no place in the entire world where some undesired activity can happen. Blockchains are structurally unable to take either approach, and so they must seek alternatives.

Fortunately, blockchains do have one very powerful tool in their grasp that makes security under such porous conditions actually feasible: *cryptography*. Cryptography allows everyone to verify that some governance procedure was executed exactly according to the rules. It leaves a verifiable evidence trail of all actions, though zero knowledge proofs allow mechanism designers freedom in picking and choosing exactly what evidence is visible and what evidence is not. Cryptography can even [prevent collusion](#)! Blockchains allow applications to live on a substrate that their governacne does not control, which allows them to effectively implement techniques such as, for example, ensure that every change to the rules only takes effect with a 60 day delay. Finally, freedom to fork is much more practical, and forking is much lower in economic and human cost, than most centralized systems.

Blockchain-based contraptions have a lot to offer the world that other kinds of systems do not. On the other hand, Nathan is completely correct to emphasize that *blockchainized* should not be equated with *financialized*. There is plenty of room for blockchain-based systems that do not look like money, and indeed we need more of them.

Alternatives to selling at below-market-clearing prices for achieving fairness (or community sentiment, or fun)

2021 Aug 22

When a seller wants to sell a fixed supply of an item that is in high (or uncertain and possibly high) demand, one choice that they often make is to set a price significantly lower than what "the market will bear". The result is that the item quickly sells out, with the lucky buyers being those who attempted to buy first. This has happened in a number of situations within the Ethereum ecosystem, notably **NFT sales** and **token sales / ICOs**. But this phenomenon is much older than that; concerts and restaurants frequently make similar choices, keeping prices cheap and leading to seats quickly selling out or buyers waiting in long lines.

Economists have for a long time asked the question: why do sellers do this? **Basic economic theory suggests that it's best if sellers sell at the market-clearing price** - that is, the price at which the amount that buyers are willing to buy exactly equals the amount the seller has to sell. **If the seller doesn't know what the market-clearing price is, the seller should sell through an auction**, and let the market determine the price. Selling below market-clearing price not only sacrifices revenue for the seller; it also can harm the *buyers*: the item may sell out so quickly that many buyers have no opportunity to get it at all, no matter how much they want it and are willing to pay to get it. Sometimes, the competitions created by these non-price-based allocation mechanisms even create negative externalities that harm third parties - an effect that, as we will see, is particularly severe in the Ethereum ecosystem.

But nevertheless, the fact that below-market-clearing pricing is so prevalent suggests that there must be some convincing reasons why sellers do it. And indeed, as the research into this topic over the last few decades has shown, there often are. And so it's worth asking the question: are there ways of achieving the same goals with more fairness, less inefficiency and less harm?

Selling at below market-clearing prices has large inefficiencies and negative externalities

If a seller sells an item at market price, or through an auction, someone who *really really wants* that item has a simple path to getting it: they can pay the high price or if it's an auction they can bid a high amount. If a seller sells the item at below market price, then demand exceeds supply, and so some people will get the item and others won't. But the mechanism deciding who will get the item is decidedly not random, and it's often not well-correlated with how much participants want the item. Sometimes, it involves being faster at clicking buttons than everyone else. At other times, it involves waking up at 2 AM in your timezone (but 11 PM or even 2 PM in someone else's). And at still other times, it just turns into an "auction by other means", one which is more chaotic, less efficient and laden with far more negative externalties.

Within the Ethereum ecosystem, there are many clear examples of this. First, we can look at the [ICO craze of 2017](). In 2017, there were a large number of projects launching initial coin offerings (ICOs), and a typical model was the **capped sale**: the project would set the price of the token and a hard maximum for how many tokens they are willing to sell, and at some point in time the sale would start automatically. Once the number of tokens hit the cap, the sale ends.

What's the result? In practice, these sales would often end in as little as 30 seconds. As soon as (or rather, just before) the sale starts, everyone would start sending transactions in to try to get in, offering higher and higher fees to encourage miners to include their transaction first. An auction by another name - except with revenues going to the miners instead of the token seller, and the extremely harmful negative externality of pricing out every other application on-chain while the sale is going on.
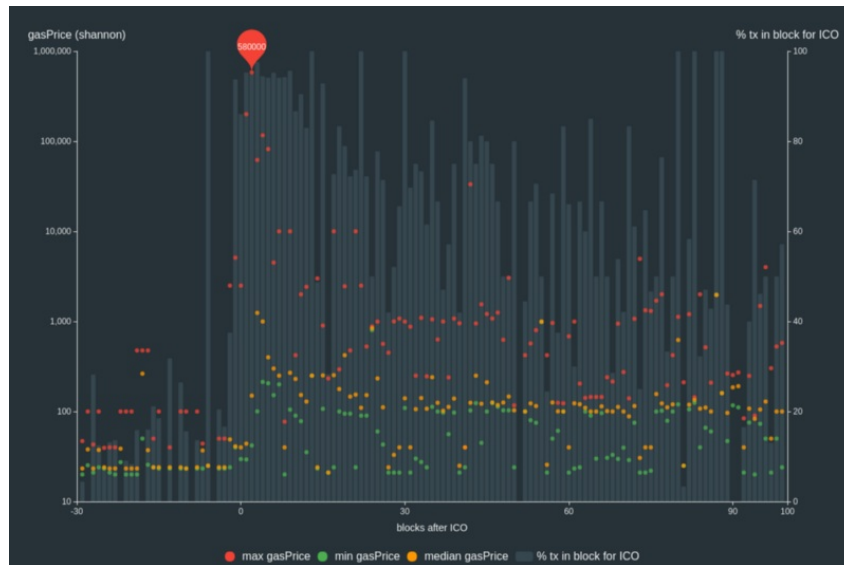
The most expensive transaction in the BAT sale set a fee of 580,000 gwei, paying a fee of $6,600 to get included in the sale.

Many ICOs after that tried various strategies to avoid these gas price auctions; one ICO notably had a smart contract that checked the transaction's gasprice and rejected it if it exceeded 50 gwei. But that of course, did not solve the problem. Buyers wishing to cheat the system sent *many* transactions, hoping that at least one would get in. Once again, an auction by another name, and this time clogging up the chain even more.

In more recent times, ICOs have become less popular, but NFTs and NFT sales are now very popular. Unfortunately, the NFT space failed to learn the lessons from 2017; they make fixed-quantity fixed-supply sales just like the ICOs did (eg. see the `mint` function on lines 97-108 of [this contract here](#)). What's the result?
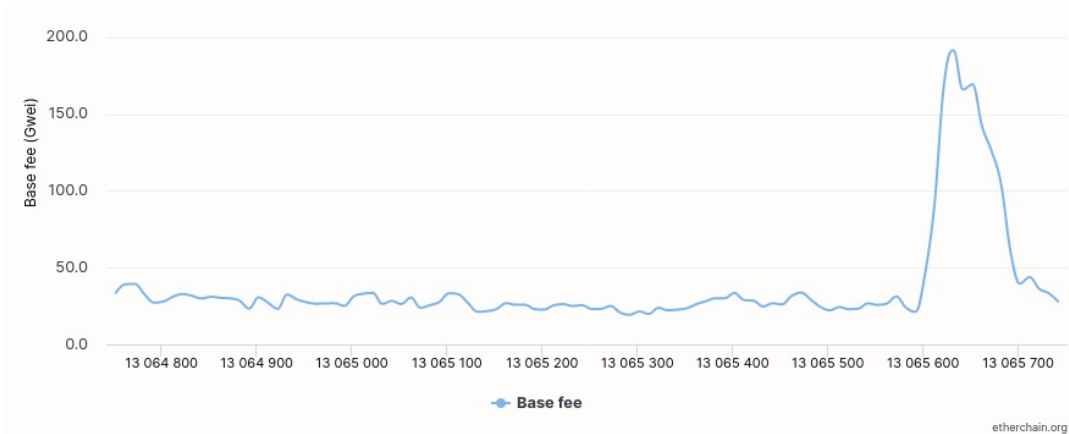
And this isn't even the biggest one; some NFT sales have created gas price spikes as high as 2000 gwei.

Once again, sky-high gas prices from users fighting each other by sending higher and higher transaction fees to get in first. An auction by another name, pricing out every other application on-chain for 15 minutes, just as before.

So why do sellers sometimes sell below market price?

Selling at below market price is hardly a new phenomenon, both within the blockchain space and

outside, and over the decades there have [been](#) many [articles](#) and [papers](#) and [podcasts](#) writing (and [sometimes](#) bitterly [complaining](#)) about the unwillingness to use auctions or set prices to market-clearing levels.

Many of the arguments are very similar between the examples in the blockchain space (NFTs and ICOs) and outside the blockchain space (popular restaurants and concerts). A particular concern is fairness and the desire to not lock poorer people out and not lose fans or create tension as a result of being perceived as greedy. Kahneman, Knetsch and Thaler's [1986 paper](#) is a good exposition of how perceptions of fairness and greed can influence these decisions. In my own recollection of the 2017 ICO season, the desire to avoid perceptions of greed was similarly a decisive factor in discouraging the use of auction-like mechanisms (I am mostly going off memory here and do not have many sources, though I *did* find a [link to a no-longer-available parody video](#) making some kind of comparison between the auction-based Gnosis ICO and the National Socialist German Workers' Party).

In addition to fairness issues, there are also the perennial arguments that products selling out and having long lines creates a perception of popularity and prestige, which makes the product seem even more attractive to others further down the line. Sure, in a rational actor model, high prices should have the same effect as long lines, but in reality long lines are much more visible than high prices are. This is just as true for ICOs and NFTs as it is for restaurants. In addition to these strategies generating more marketing value, some people actually find participating in or watching the game of grabbing up a limited set of opportunities first before everyone else takes them all to be quite fun.

But there are also some factors specific to the blockchain space. One argument for selling ICO tokens at below-market-clearing prices (and one that was decisive in convincing the OmiseGo team to adopt their capped sale strategy) has to do with community dynamics of token issuance. The most basic rule of community sentiment management is simple: you want prices to go up, not down. If community members are "in the green", they are happy. But if the price goes lower than what it was when the community members bought, leaving them at a net loss, they become unhappy and start calling you a scammer, and possibly creating a social media cascade leading to everyone else calling you a scammer.

The only way to avoid this effect is to set a sale price low enough that the post-launch market price will almost certainly be higher. But, how do you actually do this without creating a rush-for-the-gates dynamic that leads to an auction by other means?

Some more interesting solutions

The year is 2021. We have a blockchain. The blockchain contains not just a powerful decentralized finance ecosystem, but also a rapidly growing suite of all kinds of non-financial tools. The blockchain *also* presents us with a unique opportunity to reset social norms. Uber legitimized surge pricing where decades of economists yelling about "efficiency" failed; surely, blockchains can also be an

opportunity to legitimize new uses of mechanism design. And surely, instead of fiddling around with a coarse-grained one-dimensional strategy space of selling at market price versus below market price (with perhaps a second dimension for auction versus fixed-price sale), we could use our more advanced tools to create an approach that more directly solves the problems, with fewer side effects?

First, let us list the goals. We'll try to cover the cases of (i) ICOs, (ii) NFTs and (iii) conference tickets (really a type of NFT) at the same time; most of the desired properties are shared between the three cases.

1. **Fairness**: don't completely lock low-income people out of participating, give them at least some chance to get in. For token sales, there's the [not quite identical but related](#) goal of avoiding high initial wealth concentration and having a larger and more diverse initial token holder community.
2. **Don't create races**: avoid creating situations where lots of people are rushing to take the same action and only the first few get in (this is the type of situation that leads to the horrible auctions-by-another-name that we saw above).
3. **Don't require fine-grained knowledge of market conditions**: the mechanism should work even if the seller has absolutely no idea how much demand there is.
4. **Fun**: the process of participating in the sale should ideally be interesting and have game-like qualities, but without being frustrating.
5. **Give buyers positive expected returns**: in the case of a token (or, for that matter, an NFT), buyers should be more likely to see the item go up in price than go down. This necessarily implies selling to buyers at below the market price.

We can start by looking at (1). Looking at it from the point of view of Ethereum, there is a pretty clear solution. Instead of creating race conditions, just use an explicitly designed tool for the job: [proof of personhood protocols](#)! Here's one quick proposed mechanism:

> **Mechanism 1** Each participant (verified by proof-of-personhood) can buy up to X units at price P, and if they want to buy more they can buy in an auction.

It seems like it satisfies a lot of the goals already: the per-person aspect provides fairness, if the auction price turns out higher than P buyers can get positive expected returns for the portion sold through the per-person mechanism, and the auction part does not require the seller to understand the level of demand. Does it avoid creating races? If the number of participants buying through the per-person pool is not that high, it seems like it does. But what if so many people show up that the per-person pool is not big enough to provide an allocation for all of them?

Here's an idea: make the per-person allocation amount itself dynamic.

> **Mechanism 2** Each participant (verified by proof-of-personhood) can make a deposit into a smart contract to declare interest for up to X tokens. At the end, each buyer is given an allocation of min(X, N / number_of_buyers) tokens, where N is the total amount sold through the per-person pool (some other amount can also be sold by auction). The portion of the buyer's deposit going above the amount needed to buy their allocation is refunded to them.

Now, there's no race condition regardless of the number of buyers going through the per-person pool. No matter how high the demand, there's no way in which it's more beneficial to participate earlier rather than later.

Here's yet another idea, if you like your game mechanics to be more clever and use fancy quadratic formulas.

> **Mechanism 3** Each participant (verified by proof-of-personhood) can buy X units at a price $P * X^2$, up to a maximum of C tokens per buyer. C starts at some low number, and then increases over time until enough units are sold.

This mechanism has the particularly interesting property that if you're making a governance token ([please don't do that](#); this is purely harm-reduction advice), the quantity allocated to each buyer is theoretically optimal, though of course post-sale transfers will degrade this optimality over time. Mechanisms 2 and 3 seem like they both satisfy all of the above goals, at least to some extent. They're not necessarily perfect and ideal, but they *do* make good starting points.

There is one remaining issue. For fixed and limited-supply NFTs, you might get the problem that the equilibrium purchased quantity per participant is fractional (in mechanism 2, perhaps number_of_buyers > N, and in mechanism 3, perhaps setting $C = 1$ already leads to enough demand to over-subscribe the sale). In this case, you can sell fractional items by offering lottery tickets: if there are N items to be sold, then if you subscribe you have a chance of N / number_of_buyers that you

will actually get the item, and otherwise you get a refund. For a conference, groups that want to go together could be allowed to bundle their lottery tickets to guarantee either all-win or all-lose. Ability to get the item for certain can be sold at auction.

A fun mildly-grey-hat tactic for conference tickets is to disguise the pool being sold at market rate as the bottom tier of "sponsorships". You may end up with a bunch of people's faces on the sponsor board, but... maybe that's fine? After all, EthCC had John Lilic's face on their sponsor board!

In all of these cases, the core of the solution is simple: if you want to be reliably fair to *people*, then your mechanism should have some input that explicitly measures people. Proof of personhood protocols do this (and if desired can be combined with zero knowledge proofs to ensure privacy). Ergo, we should take the efficiency benefits of market and auction-based pricing, and the egalitarian benefits of proof of personhood mechanics, and combine them together.

Answers to possible questions

Q: Wouldn't lots of people who don't even care about your project buy the item through the egalitarian scheme and immediately resell it?

A: Initially, probably not. In practice, such meta-games take time to show up. But if/when they do, one possible mitigation is to make them untradeable for some period of time. This actually works because proof-of-personhood identities are untradeable: you can always use your face to claim that your previous account got hacked and the identity corresponding to you, including everything in it, should be moved to a new account.

Q: What if I want to make my item accessible not just to people in general, but to a particular community?

A: Instead of proof of personhood, use proof of participation tokens connected to events in that community. An additional alternative, also serving both egalitarian and gamification value, is to lock some items inside solutions to some publicly-published puzzles.

Q: How do we know people will accept this? People have been resistant to weird new mechanisms in the past.

A: It's very difficult to get people to accept a new mechanism that they find weird by having economists write screeds about how they "should" accept it for the sake of "efficiency" (or even "equity"). However, rapid changes in context do an excellent job of resetting people's set expectations. So if there's any good time at all to try this, the blockchain space is that time. You could also wait for the "metaverse", but it's quite possible that the best version of the metaverse will run on Ethereum anyway, so you might as well just start now.

Moving beyond coin voting governance

2021 Aug 16

Special thanks to Karl Floersch, Dan Robinson and Tina Zhen for feedback and review. See also [Notes on Blockchain Governance](), [Governance, Part 2: Plutocracy Is Still Bad](), [On Collusion]() and [Coordination, Good and Bad]() for earlier thinking on similar topics.

One of the important trends in the blockchain space over the past year is the transition from focusing on **decentralized finance (DeFi)** to also thinking about **decentralized governance (DeGov)**. While the 2020 is [often]() widely, and with much justification, [hailed]() as [a year of DeFi](), over the year since then the growing complexity and capability of DeFi projects that make up this trend has led to growing interest in decentralized governance to handle that complexity. There are examples inside of Ethereum: [YFI](), [Compound](), [Synthetix](), [UNI](), [Gitcoin]() and others have all launched, or even *started with*, some kind of DAO. But it's also true outside of Ethereum, with arguments over [infrastructure]() [funding]() [proposals]() in Bitcoin Cash, [infrastructure funding votes in Zcash](), and much more.

The rising popularity of formalized decentralized governance of some form is undeniable, and there are important reasons why people are interested in it. But it is also important to keep in mind the risks of such schemes, as the recent [hostile takeover of Steem]() and subsequent mass exodus to Hive makes clear. I would further argue that these trends are unavoidable. **Decentralized governance in some contexts is both necessary and dangerous**, for reasons that I will get into in this post. How can we get the benefits of DeGov while minimizing the risks? I will argue for one key part of the answer: **we need to move beyond coin voting as it exists in its present form**.

DeGov is necessary

Ever since the [Declaration of Independence of Cyberspace]() in 1996, there has been a key unresolved contradiction in what can be called cypherpunk ideology. On the one hand, cypherpunk values are all about using cryptography to minimize coercion, and maximize the efficiency and reach of the main non-coercive coordination mechanism available at the time: private property and markets. On the other hand, the economic logic of private property and markets is optimized for activities that [can be "decomposed"]() into repeated one-to-one interactions, and the infosphere, where art, documentation, science and code are produced and consumed through irreducibly one-to-many interactions, is the exact opposite of that.

There are two key problems inherent to such an environment that need to be solved:

- **Funding public goods**: how do projects that are valuable to a wide and unselective group of people in the community, but which often do not have a business model (eg. layer-1 and layer-2 protocol research, client development, documentation...), get funded?
- **Protocol maintenance and upgrades**: how are upgrades to the protocol, and regular maintenance and adjustment operations on parts of the protocol that are not long-term stable (eg. lists of safe assets, price oracle sources, multi-party computation keyholders), agreed upon?

Early blockchain projects largely ignored both of these challenges, pretending that the only public good that mattered was network security, which could be achieved with a single algorithm set in stone forever and paid for with fixed proof of work rewards. This state of affairs in funding was possible at first because of extreme Bitcoin price rises from 2010-13, then the one-time ICO boom from 2014-17, and again from the simultaneous second crypto bubble of 2014-17, all of which made the ecosystem wealthy enough to temporarily paper over the large market inefficiencies. Long-term governance of public resources was similarly ignored: Bitcoin took the path of extreme minimization, focusing on providing a fixed-supply currency and ensuring support for layer-2 payment systems like Lightning and nothing else, Ethereum continued developing mostly harmoniously (with [one major exception]()) because of the strong [legitimacy]() of its pre-existing roadmap (basically: "proof of stake and sharding"), and sophisticated application-layer projects that required anything more did not yet exist.

But now, increasingly, that luck is running out, and challenges of coordinating protocol maintenance and upgrades and funding documentation, research and development while avoiding the risks of centralization are at the forefront.

The need for DeGov for funding public goods

It is worth stepping back and seeing the absurdity of the present situation. Daily mining issuance rewards from Ethereum are about 13500 ETH, or about $40m, [per day](). [Transaction fees]() are similarly high; the [non-EIP-1559-burned portion]() continues to be around 1,500 ETH (~$4.5m) per day. So there are many billions of dollars per year going to fund network security. Now, what is the budget of the Ethereum Foundation? About $30-60 million per year. There are non-EF actors (eg. Consensys) contributing to development, but they are not much larger. The situation in Bitcoin is similar, with perhaps even less funding going into non-security public goods.

Here is the situation in a chart:

Estimated Bitcoin and Ethereum spending on PoW vs R&D
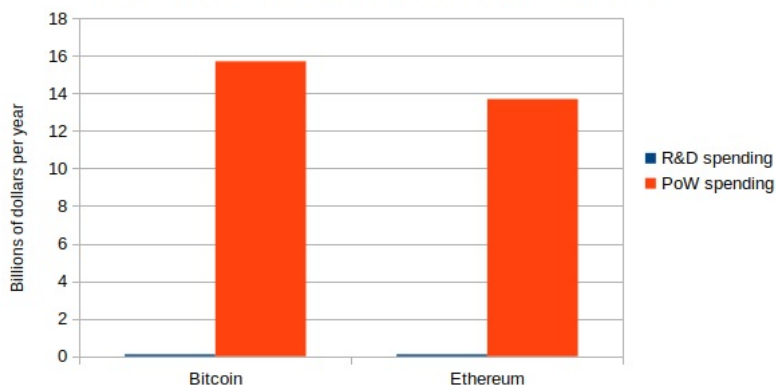
Within the Ethereum ecosystem, one can make a case that this disparity does not matter too much; tens of millions of dollars per year is "enough" to do the needed R&D and adding more funds does not necessarily improve things, and so the risks to the platform's credible neutrality from instituting in-protocol developer funding exceed the benefits. But in many smaller ecosystems, both ecosystems within Ethereum and entirely separate blockchains like BCH and Zcash, the same debate is brewing, and at those smaller scales the imbalance makes a big difference.

Enter DAOs. A project that launches as a "pure" DAO from day 1 can achieve a combination of two properties that were previously impossible to combine: (i) sufficiency of developer funding, and (ii) credible neutrality of funding (the much-coveted "fair launch"). Instead of developer funding coming from a hardcoded list of receiving addresses, the decisions can be made by the DAO itself.
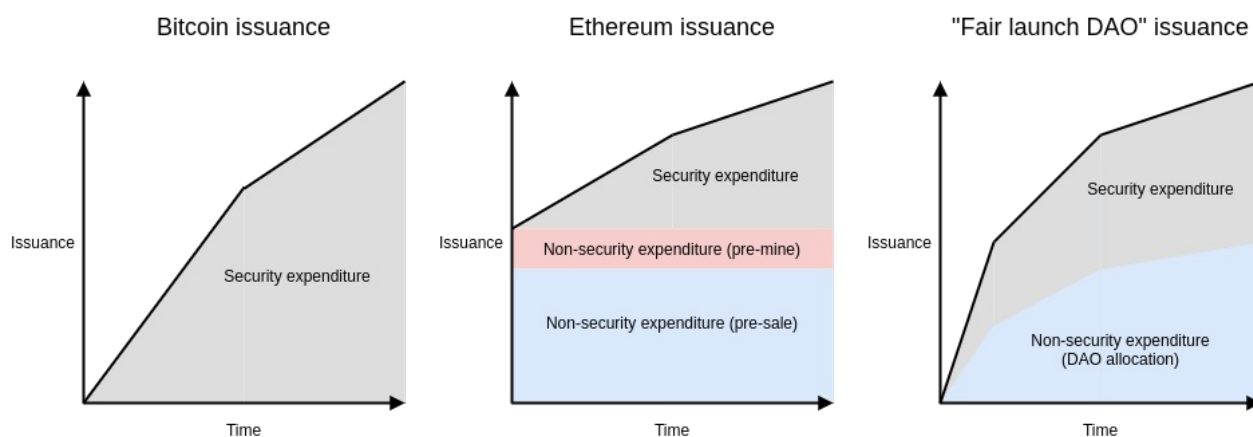
Of course, it's difficult to make a launch perfectly fair, and unfairness from information asymmetry can often be worse than unfairness from explicit premines (was Bitcoin *really* a fair launch considering how few people had a chance to even hear about it by the time 1/4 of the supply had already been handed out by the end of 2010?). But even still, in-protocol compensation for non-security public goods from day one seems like a potentially significant step forward toward getting sufficient and more credibly neutral developer funding.

The need for DeGov for protocol maintenance and upgrades

In addition to public goods funding, the other equally important problem requiring governance is protocol maintenance and upgrades. While I advocate trying to minimize all non-automated parameter adjustment (see the "limited governance" section below) and I am a fan of RAI's "un-governance" strategy, there are times where governance is unavoidable. Price oracle inputs must come from somewhere, and occasionally that somewhere needs to change. Until a protocol "ossifies" into its final form, improvements have to be coordinated somehow. Sometimes, a protocol's community might *think* that they are ready to ossify, but then the world throws a curveball that requires a complete and controversial restructuring. What happens if the US dollar collapses, and RAI has to scramble to create and maintain their own decentralized CPI index for their stablecoin to remain stable and relevant? Here too, DeGov is necessary, and so avoiding it outright is not a viable solution.

One important distinction is whether or not *off-chain* governance is possible. I have for a long time been a fan of off-chain governance wherever possible. And indeed, for base-layer blockchains, off-chain governance absolutely *is* possible. **But for application-layer projects, and especially defi projects, we run into the problem that application-layer smart contract systems often *directly control external assets*, and that control cannot be forked away.** If Tezos's on-chain governance gets captured by an attacker, the community can hard-fork away without any losses beyond (admittedly high) coordination costs. If MakerDAO's on-chain governance gets captured

by an attacker, the community can absolutely spin up a new MakerDAO, but they will lose all the ETH and other assets that are stuck in the existing MakerDAO CDPs. **Hence, while off-chain governance is a good solution for base layers and *some* application-layer projects, many application-layer projects, particularly DeFi, will inevitably require formalized on-chain governance of some form.**

DeGov is dangerous

However, all current instantiations of decentralized governance come with great risks. To followers of my writing, this discussion will not be new; the risks are much the same as those that I talked about [here](#), [here](#) and [here](#). There are two primary types of issues with coin voting that I worry about: **(i) inequalities and incentive misalignments even in the absence of attackers, and (ii) outright attacks through various forms of (often obfuscated) vote buying.** To the former, there have already been many proposed mitigations (eg. delegation), and there will be more. But the latter is a much more dangerous elephant in the room to which I see no solution within the current coin voting paradigm.

Problems with coin voting even in the absence of attackers

The problems with coin voting even without explicit attackers are increasingly well-understood (eg. see [this recent piece by DappRadar and Monday Capital](#)), and mostly fall into a few buckets:

- **Small groups of wealthy participants ("whales") are better at successfully executing decisions than large groups of small-holders**. This is because of the tragedy of the commons among small-holders: each small-holder has only an insignificant influence on the outcome, and so they have little incentive to not be lazy and actually vote. Even if there are rewards for voting, there is little incentive to research and think carefully about what they are voting for.
- **Coin voting governance empowers coin holders and coin holder interests at the expense of other parts of the community**: protocol communities are made up of diverse constituencies that have many different values, visions and goals. Coin voting, however, only gives power to one constituency (coin holders, and especially wealthy ones), and leads to over-valuing the goal of making the coin price go up even if that involves harmful rent extraction.
- **Conflict of interest issues**: giving voting power to one constituency (coin holders), and especially over-empowering wealthy actors in that constituency, risks over-exposure to the conflicts-of-interest within that particular elite (eg. investment funds or holders that *also* hold tokens of other DeFi platforms that interact with the platform in question)

There is one major type of strategy being attempted for solving the first problem (and therefore also mitigating the third problem): [delegation](#). Smallholders don't have to personally judge each decision; instead, they can delegate to community members that they trust. This is an honorable and worthy experiment; we shall see how well delegation can mitigate the problem.
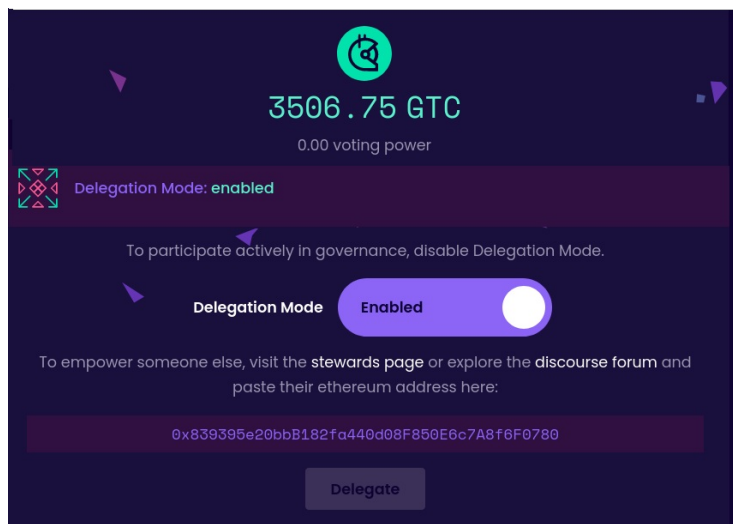
*My voting delegation page in the Gitcoin DAO*

The problem of coin holder centrism, on the other hand, is significantly more challenging: coin holder centrism is inherently baked into a system where coin holder votes are the only input. The mis-perception that coin holder centrism is an intended goal, and not a bug, is already causing confusion and harm; one (broadly excellent) [article discussing blockchain public goods](#) complains:

> Can crypto protocols be considered public goods if ownership is concentrated in the hands of a few whales? Colloquially, these market primitives are sometimes described as "public infrastructure," but if blockchains serve a "public" today, it is primarily one of decentralized finance. Fundamentally, these tokenholders share only one common object of concern: price.

The complaint is false; blockchains serve a public much richer and broader than DeFi token holders. But our coin-voting-driven governance systems are completely failing to capture that, and it seems difficult to make a governance system that captures that richness without a more fundamental change to the paradigm.

Coin voting's deep fundamental vulnerability to attackers: vote buying

The problems get much worse once determined attackers trying to subvert the system enter the picture. The fundamental vulnerability of coin voting is simple to understand. **A token in a protocol with coin voting is a bundle of two rights that are combined into a single asset: (i) some kind of economic interest in the protocol's revenue and (ii) the right to participate in governance. This combination is deliberate: the goal is to align power and responsibility. But in fact, these two rights are very easy to unbundle from each other**. Imagine a simple wrapper contract that has these rules: if you deposit 1 XYZ into the contract, you get back 1 WXYZ. That WXYZ can be converted back into an XYZ at any time, plus in addition it accrues dividends. Where do the dividends come from? Well, while the XYZ coins are inside the wrapper contract, it's the wrapper contract that has the ability to use them however it wants in governance (making proposals, voting on proposals, etc). The wrapper contract simply auctions off this right every day, and distributes the profits among the original depositors.
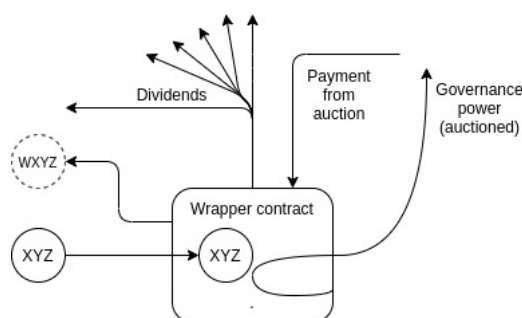
As an XYZ holder, is it in your interest to deposit your coins into the contract? If you are a very large holder, it might not be; you like the dividends, but you are scared of what a misaligned actor might do with the governance power you are selling them. But if you are a smaller holder, then it very much is. If the governance power auctioned by the wrapper contract gets bought up by an attacker, you personally only suffer a small fraction of the cost of the bad governance decisions that your token is contributing to, but you personally gain the full benefit of the dividend from the governance rights auction. This situation is a classic tragedy of the commons.

Suppose that an attacker makes a decision that corrupts the DAO to the attacker's benefit. The harm per participant from the decision succeeding is D, and the chance that a single vote tilts the outcome is p. Suppose an attacker makes a bribe of B. The game chart looks like this:

Decision	Benefit to you	Benefit to others
Accept attacker's bribe	$B - D * p$	$-999 * D * p$
Reject bribe, vote your conscience	0	0

If $B > D * p$, you are inclined to accept the bribe, but as long as $B < 1000 * D * p$, accepting the bribe is *collectively* harmful. So if $p < 1$ (usually, p is far below 1), there is an opportunity for an attacker to bribe users to adopt a net-negative decision, compensating each user far less than the harm they suffer.

One natural critique of voter bribing fears is: are voters *really* going to be so immoral as to accept such obvious bribes? The average DAO token holder is an enthusiast, and it would be hard for them to feel good about so selfishly and blatantly selling out the project. But what this misses is that there are much more obfuscated ways to separate out profit sharing rights from governance rights, that don't require anything remotely as explicit as a wrapper contract.

The simplest example is borrowing from a defi lending platform (eg. [Compound](#)). Someone who already holds ETH can lock up their ETH in a CDP ("collateralized debt position") in one of these platforms, and once they do that the CDP contract allows them to borrow an amount of XYZ up to eg. half the value of the ETH that they put in. They can then do whatever they want with this XYZ. To recover their ETH, they would eventually need to pay back the XYZ that they borrowed, plus interest.
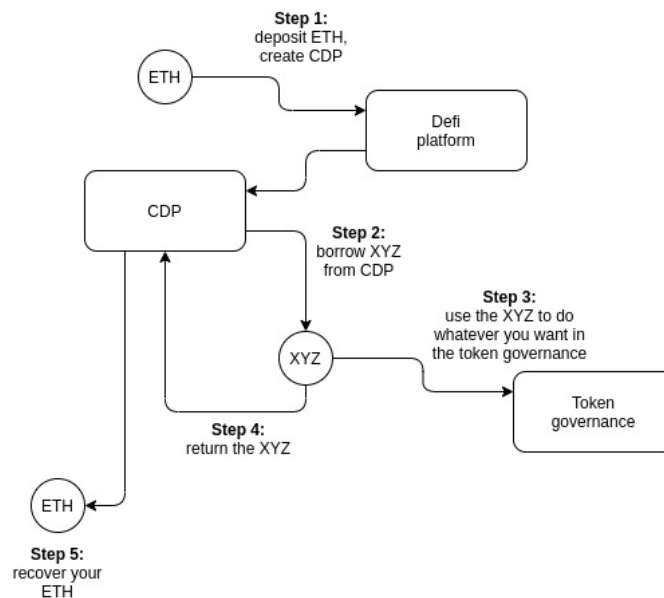
Note that throughout this process, *the borrower has no financial exposure to XYZ*. That is, if they use their XYZ to vote for a governance decision that destroys the value of XYZ, they do not lose a penny as a result. The XYZ they are holding is XYZ that they have to eventually pay back into the CDP regardless, so they do not care if its value goes up or down. **And so we have achieved unbundling: the borrower has governance power without economic interest, and the lender has economic interest without governance power**.

There are also centralized mechanisms for separating profit sharing rights from governance rights. Most notably, when users deposit their coins on a (centralized) exchange, the exchange holds full custody of those coins, and the exchange has the ability to use those coins to vote. This is not mere theory; there is evidence of exchanges using their users' coins in several DPoS systems. The most notable recent example is the [attempted hostile takeover of Steem](), where exchanges used their customers' coins to vote for some proposals that helped to cement a takeover of the Steem network that the bulk of the community strongly opposed. The situation was only resolved through an outright mass exodus, where a large portion of the community moved to a different chain called [Hive]().

Some DAO protocols are using timelock techniques to limit these attacks, requiring users to lock their coins and make them immovable for some period of time in order to vote. These techniques can limit buy-then-vote-then-sell attacks in the short term, but ultimately [timelock mechanisms can be bypassed]() by users holding and voting with their coins through a contract that issues a wrapped version of the token (or, more trivially, a centralized exchange). **As far as security mechanisms go, timelocks are more like a paywall on a newspaper website than they are like a lock and key.**

At present, many blockchains and DAOs with coin voting have so far managed to avoid these attacks in their most severe forms. There are occasional signs of attempted bribes:

But despite all of these important issues, there have been much fewer examples of outright voter bribing, including obfuscated forms such as using financial markets, that simple economic reasoning would suggest. The natural question to ask is: why haven't more outright attacks happened yet?

My answer is that the "why not yet" relies on three contingent factors that are true today, but are likely to get less true over time:

1. **Community spirit** from having a tightly-knit community, where everyone feels a sense of camaraderie in a common tribe and mission..
2. **High wealth concentration and coordination of token holders**; large holders have higher ability to affect the outcome and have investments in long-term relationships with each other (both the "old boys clubs" of VCs, but also many other equally powerful but lower-profile groups of wealthy token holders), and this makes them much more difficult to bribe.
3. **Immature financial markets** in governance tokens: ready-made tools for making wrapper tokens exist in proof-of-concept forms but are not widely used, bribing contracts exist but are similarly immature, and liquidity in lending markets is low.

When a small coordinated group of users holds over 50% of the coins, *and* both they and the rest are invested in a tightly-knit community, and there are few tokens being lent out at reasonable rates, all of the above bribing attacks may perhaps remain theoretical. But over time, (1) and (3) will inevitably become less true no matter what we do, and (2) *must* become less true if we want DAOs to become more fair. When those changes happen, will DAOs remain safe? And if coin voting cannot be sustainably resistant against attacks, then what can?

Solution 1: limited governance

One possible mitigation to the above issues, and one that is to varying extents being tried already, is to put limits on what coin-driven governance can do. There are a few ways to do this:

- **Use on-chain governance only for applications, not base layers**: Ethereum does this already, as the protocol itself is governed through off-chain governance, while DAOs and other apps on top of this are sometimes (but not always) governed through on-chain governance
- **Limit governance to fixed parameter choices**: Uniswap does this, as it only allows governance to affect (i) token distribution and (ii) a 0.05% fee in the Uniswap exchange. Another great example is RAI's "un-governance" roadmap, where governance has control over fewer and fewer features over time.
- **Add time delays**: a governance decision made at time T only takes effect at eg. T + 90 days. This allows users and applications that consider the decision unacceptable to move to another application (possibly a fork). Compound has a time delay mechanism in its governance, but in principle the delay can (and eventually should) be much longer.
- **Be more fork-friendly**: make it easier for users to quickly coordinate on and execute a fork. This makes the payoff of capturing governance smaller.

The Uniswap case is particularly interesting: it's an intended behavior that the on-chain governance funds teams, which may develop future versions of the Uniswap protocol, but *it's up to users to opt-in* to upgrading to those versions. This is a hybrid of on-chain and off-chain governance that leaves only a limited role for the on-chain side.

But limited governance is not an acceptable solution by itself; those areas where governance is needed the most (eg.

funds distribution for public goods) are themselves among the most vulnerable to attack. Public goods funding is so vulnerable to attack because there is a very direct way for an attacker to profit from bad decisions: they can try to push through a bad decision that sends funds to themselves. Hence, we also need techniques to improve governance itself...

Solution 2: non-coin-driven governance

A second approach is to use forms of governance that are not coin-voting-driven. But if coins do not determine what weight an account has in governance, what does? There are two natural alternatives:

1. **Proof of personhood systems**: systems that verify that accounts correspond to unique individual humans, so that governance can assign one vote per human. See [here](#) for a review of some techniques being developed, and [ProofOfHumanity](#) and [BrightID](#) and [Idenanetwork](#) for three attempts to implement this.
2. **Proof of participation**: systems that attest to the fact that some account corresponds to a person that has participated in some event, passed some educational training, or performed some useful work in the ecosystem. See [POAP](#) for one attempt to implement thus.

There are also hybrid possibilities: one example is [quadratic voting](#), which makes the power of a single voter proportional to the *square root* of the economic resources that they commit to a decision. Preventing people from gaming the system by splitting their resource across many identities requires proof of personhood, and the still-existent financial component allows participants to credibly signal how strongly they care about an issue, as well as how strongly they care about the ecosystem. Gitcoin quadratic funding is a form of quadratic voting, and quadratic voting DAOs [are being built](#).

Proof of participation is less well-understood. The key challenge is that determining what counts as how much participation itself requires a quite robust governance structure. It's possible that the easiest solution involves bootstrapping the system with a hand-picked choice of 10-100 early contributors, and then decentralizing over time as the selected participants of round N determine participation criteria for round N+1. The possibility of a fork helps provide a path to recovery from, and an incentive against, governance going off the rails.

Proof of personhood and proof of participation both require some form of anti-collusion (see [article explaining the issue here](#) and [MACI documentation here](#)) to ensure that the non-money resource being used to measure voting power remains non-financial, and does not itself end up inside of smart contracts that sell the governance power to the highest bidder.

Solution 3: skin in the game

The third approach is to break the tragedy of the commons, by changing the rules of the vote itself. **Coin voting fails because while voters are *collectively* accountable for their decisions (if everyone votes for a terrible decision, everyone's coins drop to zero), each voter is not *individually* accountable (if a terrible decision happens, those who supported it suffer no more than those who opposed it). Can we make a voting system that changes this dynamic, and makes voters individually, and not just collectively, responsible for their decisions?**

Fork-friendliness is arguably a skin-in-the-game strategy, if forks are done in the way that Hive forked from Steem. In the case that a ruinous governance decision succeeds and can no longer be opposed inside the protocol, users can take it upon themselves to make a fork. Furthermore, in that fork, the coins that voted for the bad decision can be destroyed.

This sounds harsh, and perhaps it even feels like a violation of an implicit norm that the "immutability of the ledger" should remain sacrosanct when forking a coin. But the idea seems much more reasonable when seen from a

different perspective. We keep the idea of a strong firewall where individual coin balances are expected to be inviolate, *but only apply that protection to coins that do not participate in governance*. If you participate in governance, even indirectly by putting your coins into a wrapper mechanism, then you may be held liable for the costs of your actions.

This creates individual responsibility: if an attack happens, and your coins vote for the attack, then your coins are destroyed. If your coins do not vote for the attack, your coins are safe. The responsibility propagates upward: if you put your coins into a wrapper contract and the wrapper contract votes for an attack, the wrapper contract's balance is wiped and so you lose your coins. If an attacker borrows XYZ from a defi lending platform, when the platform forks anyone who lent XYZ loses out (note that this makes lending the governance token in general very risky; this is an intended consequence).

Skin-in-the-game in day-to-day voting

But the above only works for guarding against decisions that are truly extreme. What about smaller-scale heists, which unfairly favor attackers manipulating the economics of the governance but not severely enough to be ruinous? And what about, in the absence of any attackers at all, simple laziness, and the fact that coin-voting governance has no selection pressure in favor of higher-quality opinions?

The most popular solution to these kinds of issues is [futarchy](#), introduced by Robin Hanson in the early 2000s. Votes become bets: to vote in favor of a proposal, you make a bet that the proposal will lead to a good outcome, and to vote against the proposal, you make a bet that the proposal will lead to a poor outcome. Futarchy introduces individual responsibility for obvious reasons: if you make good bets, you get more coins, and if you make bad bets you lose your coins.
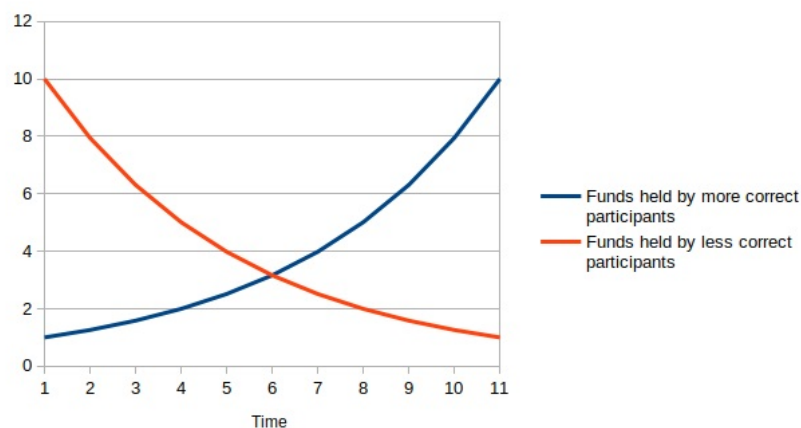
"Pure" futarchy has proven difficult to introduce, because in practice objective functions are very difficult to define (it's not just coin price that people want!), but various hybrid forms of futarchy may well work. Examples of hybrid futarchy include:

- **Votes as buy orders**: see [ethresear.ch post](#). Voting in favor of a proposal requires making an enforceable buy order to buy additional tokens at a price somewhat lower than the token's current price. This ensures that if a terrible decision succeeds, those who support it may be forced to buy their opponents out, but it also ensures that in more "normal" decisions coin holders have more slack to decide according to non-price criteria if they so wish.
- **Retroactive public goods funding**: see [post with the Optimism team](#). Public goods are funded by some voting mechanism *retroactively*, after they have already achieved a result. Users can buy *project tokens* to fund their project while signaling confidence in it; buyers of project tokens get a share of the reward if that project is deemed to have achieved a desired goal.
- **Escalation games**: see [Augur](#) and [Kleros](#). Value-alignment on lower-level decisions is incentivized by the possibility to appeal to a higher-effort but higher-accuracy higher-level process; voters whose votes agree with the ultimate decision are rewarded.

In the latter two cases, hybrid futarchy depends on some form of non-futarchy governance to measure against the objective function or serve as a dispute layer of last resort. However, this non-futarchy governance has several advantages that it does not if used directly: (i) it activates later, so it has access to more information, (ii) it is used less frequently, so it can expend less effort, and (iii) each use of it has greater consequences, so it's more acceptable to just rely on forking to align incentives for this final layer.

Hybrid solutions

There are also solutions that combine elements of the above techniques. Some possible examples:

- **Time delays plus elected-specialist governance**: this is one possible solution to the ancient conundrum of how to make an crypto-collateralized stablecoin whose locked funds can exceed the value of the profit-taking

token without risking governance capture. The stable coin uses a price oracle constructed from the median of values submitted by N (eg. N = 13) elected providers. Coin voting chooses the providers, but it can only cycle out one provider each week. If users notice that coin voting is bringing in untrustworthy price providers, they have N/2 weeks before the stablecoin breaks to switch to a different one.

- **Futarchy + anti-collusion = reputation**: Users vote with "reputation", a token that cannot be transferred. Users gain more reputation if their decisions lead to desired results, and lose reputation if their decisions lead to undesired results. See here for an article advocating for a reputation-based scheme.
- **Loosely-coupled (advisory) coin votes**: a coin vote does not directly implement a proposed change, instead it simply exists to make its outcome public, to build legitimacy for off-chain governance to implement that change. This can provide the benefits of coin votes, with fewer risks, as the legitimacy of a coin vote drops off automatically if evidence emerges that the coin vote was bribed or otherwise manipulated.

But these are all only a few possible examples. There is much more that can be done in researching and developing non-coin-driven governance algorithms. **The most important thing that can be done today is moving away from the idea that coin voting is the only legitimate form of governance decentralization**. Coin voting is attractive because it *feels* credibly neutral: anyone can go and get some units of the governance token on Uniswap. In practice, however, **coin voting may well only appear secure today precisely because of the imperfections in its neutrality** (namely, large portions of the supply staying in the hands of a tightly-coordinated clique of insiders).

We should stay very wary of the idea that current forms of coin voting are "safe defaults". There is still much that remains to be seen about how they function under conditions of more economic stress and mature ecosystems and financial markets, and the time is now to start simultaneously experimenting with alternatives.

Against overuse of the Gini coefficient

2021 Jul 29

Special thanks to Barnabe Monnot and Tina Zhen for feedback and review

The [Gini coefficient](#) (also called the Gini index) is by far the most popular and widely known measure of inequality, typically used to measure inequality of income or wealth in some country, territory or other community. It's popular because it's easy to understand, with a mathematical definition that can easily be visualized on a graph.

However, as one might expect from any scheme that tried to reduce inequality to a single number, the Gini coefficient also has its limits. This is true even in its original context of measuring income and wealth inequality in countries, but it becomes even more true when the Gini coefficient is transplanted into other contexts (particularly: cryptocurrency). In this post I will talk about some of the limits of the Gini coefficient, and propose some alternatives.

What is the Gini coefficient?

The [Gini coefficient](#) is a measure of inequality introduced by Corrado Gini in 1912. It is typically used to measure inequality of income and wealth of countries, though it is also increasingly being used in other contexts.

There are two equivalent definitions of the Gini coefficient:

Area-above-curve definition: draw the graph of a function, where $f(p)$ equals the share of total income earned by the lowest-earning portion of the population (eg. $f(0.1)$ is the share of total income earned by the lowest-earning 10%). The Gini coefficient is the area between that curve and the $y=x$ line, as a portion of the whole triangle:
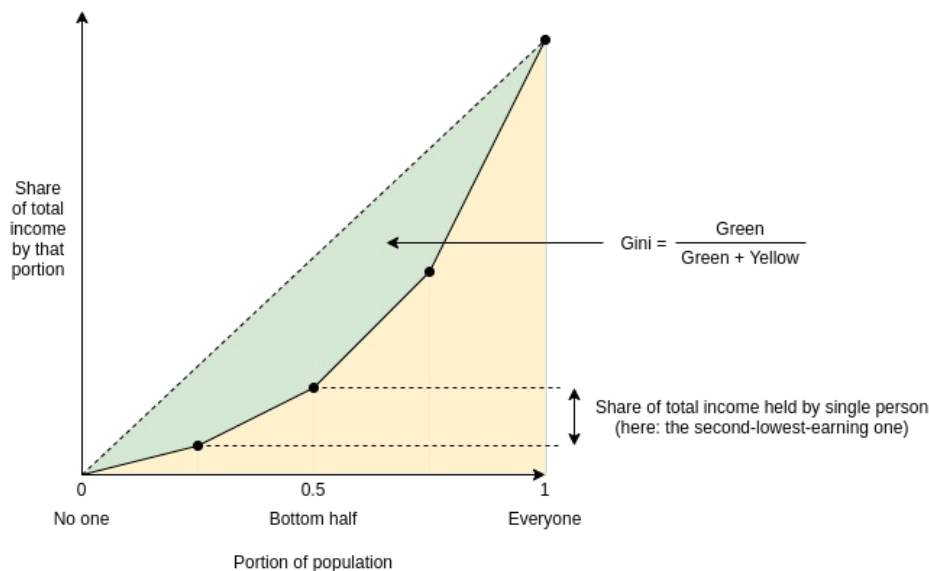
Average-difference definition: the Gini coefficient is half the average difference of incomes between each all possible pairs of individuals, divided by the mean income.

For example, in the above example chart, the four incomes are `[1, 2, 4, 8]`, so the 16 possible differences are `[0, 1, 3, 7, 1, 0, 2, 6, 3, 2, 0, 4, 7, 6, 4, 0]`. Hence the average difference is 2.875 and the mean income is 3.75, so Gini = $\frac{2.875}{2 * 3.75} \approx 0.3833$.

It turns out that the two are mathematically equivalent (proving this is an exercise to the reader)!

What's wrong with the Gini coefficient?

The Gini coefficient is attractive because it's a reasonably simple and easy-to-understand statistic. It might not *look* simple, but trust me, pretty much everything in statistics that deals with populations of arbitrary size is that bad, and often much worse. Here, stare at the formula of something as basic as the [standard deviation](#):

$$\sigma = \frac{\sum_{i=1}^n x_i^2}{n} - (\frac{\sum_{i=1}^n x_i}{n})^2$$

And here's the Gini:

$$G = \frac{2 * \sum_{i=1}^n i*x_i}{n * \sum_{i=1}^n x_i} - \frac{n+1}{n}$$

It's actually quite tame, I promise!

So, what's wrong with it? Well, there are lots of things wrong with it, and people have [written](#) lots of [articles](#) about [various problems](#) with the Gini coefficient. In this article, I will focus on one specific problem that I think is under-discussed about the Gini as a whole, but that has particular relevance to analyzing inequality in internet communities such as blockchains. **The Gini coefficient combines together into a single inequality index two problems that actually look quite different: suffering due to lack of resources and concentration of power**.

To understand the difference between the two problems more clearly, let's look at two dystopias:
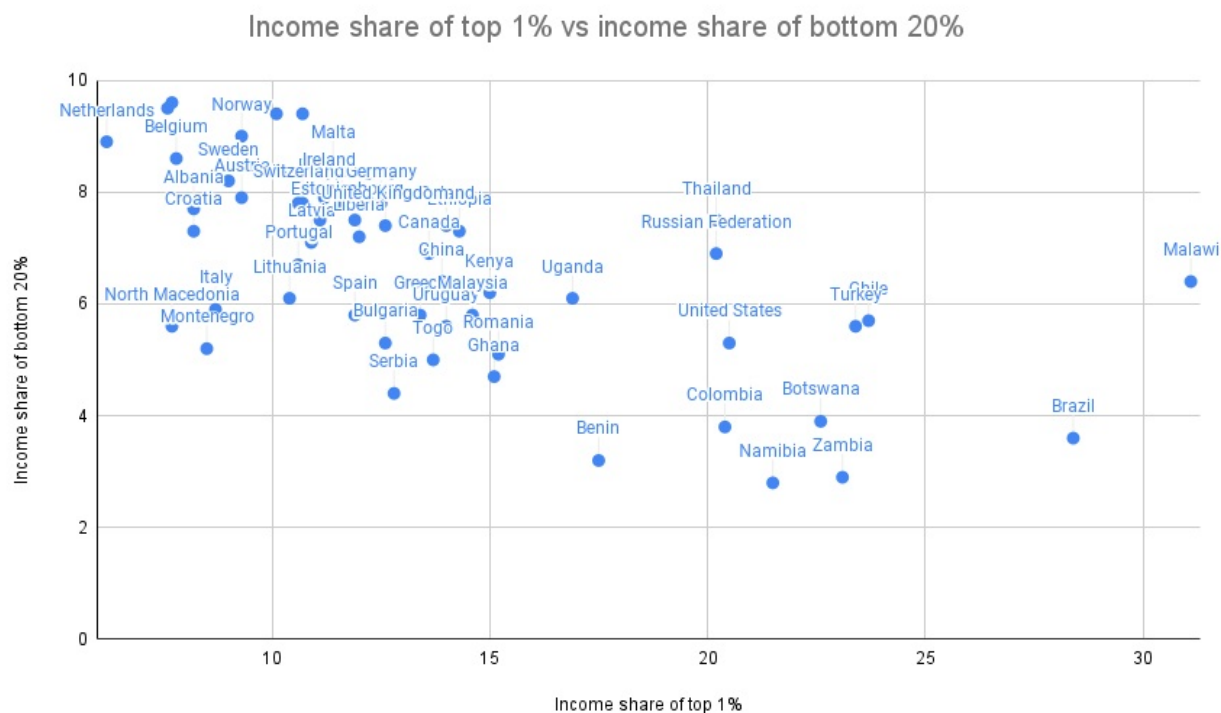
- **Dystopia A**: half the population equally shares all the resources, everyone else has none
- **Dystopia B**: one person has half of all the resources, everyone else equally shares the remaining half

Here are the Lorenz curves (fancy charts like we saw above) for both dystopias:
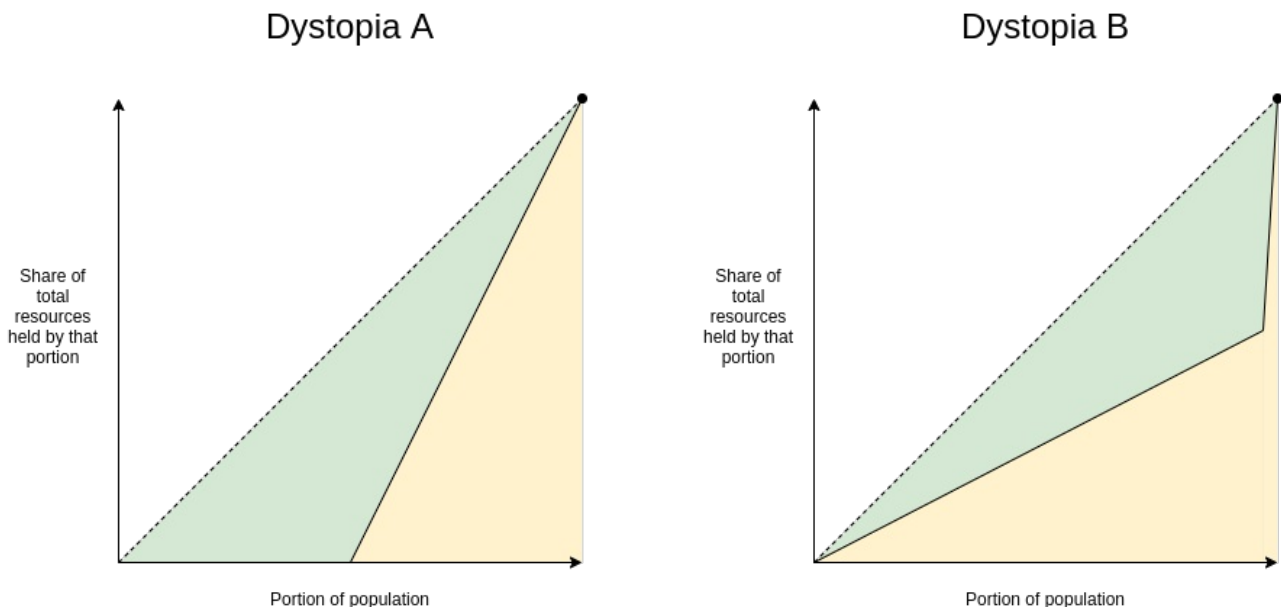
Clearly, neither of those two dystopias are good places to live. **But they are not-very-nice places to live in very different ways**. Dystopia A gives each resident a coin flip between unthinkably horrific mass starvation if they end up on the left half on the distribution and egalitarian harmony if they end up on the right half. If you're [Thanos](#), you might actually like it! If you're not, it's worth avoiding with the strongest force. Dystopia B, on the other hand, is Brave New World-like: everyone has decently good lives (at least at the time when that snapshot of everyone's resources is taken), but at the high cost of an extremely undemocratic power structure where you'd better hope you have a good overlord. If you're [Curtis Yarvin](#), you might actually like it! If you're not, it's very much worth avoiding too.

These two problems are different enough that they're worth analyzing and measuring separately. And this difference is not just theoretical. Here is a chart showing share of total income earned by the bottom 20% (a decent proxy for avoiding dystopia A) versus share of total income earned by the top 1% (a decent proxy for being near dystopia B):

Income share of top 1% vs income share of bottom 20%

Sources: https://data.worldbank.org/indicator/SI.DST.FRST.20 (merging 2015 and 2016 data) and http://hdr.undp.org/en/indicators/186106.

The two are clearly correlated (coefficient -0.62), but very far from perfectly correlated (the high priests of statistics apparently consider 0.7 to be the lower threshold for being "highly correlated", and we're even under that). There's an interesting second dimension to the chart that can be analyzed - what's the difference between a country where the top 1% earn 20% of the total income and the bottom 20% earn 3% and a country where the top 1% earn 20% and the bottom 20% earn 7%? Alas, such an exploration is best left to other enterprising data and culture explorers with more experience than myself.

Why Gini is *very* problematic in non-geographic communities (eg. internet/crypto communities)

Wealth concentration within the blockchain space in particular is an important problem, and it's a problem worth measuring and understanding. It's important for the blockchain space as a whole, as many people (and US senate hearings) are trying to figure out to what extent crypto is truly anti-elitist and to what extent it's just replacing old elites with new ones. It's also important when comparing different cryptocurrencies with each other.
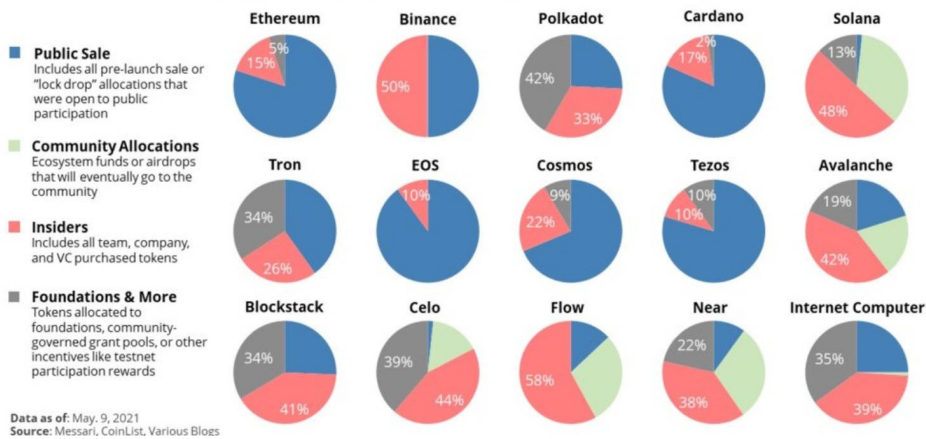
Share of coins explicitly allocated to specific insiders in a cryptocurrency's initial supply is one type of inequality. Note that the Ethereum data is slightly wrong: the insider and foundation shares should be 12.3% and 4.2%, not 15% and 5%.

Given the level of concern about these issues, it should be not at all surprising that many people have tried computing Gini indices of cryptocurrencies:

- The observed Gini index for staked EOS tokens (2018)
- Gini coefficients of cryptocurrencies (2018)
- Measuring decentralization in Bitcoin and Ethereum using Multiple Metrics and Granularities (2021, includes Gini and 2 other metrics)
- Nouriel Roubini comparing Bitcoin's Gini to North Korea (2018)
- On-chain Insights in the Cryptocurrency Markets (2021, uses Gini to measure concentration)

And even earlier than that, we had to deal with this sensationalist article from 2014:

In addition to common plain methodological mistakes (often either mixing up income vs wealth inequality, mixing up users vs accounts, or both) that such analyses make quite frequently, there is a deep and subtle problem with using the Gini coefficient to make these kinds of comparisons. The problem lies in key distinction between typical geographic communities (eg. cities, countries) and typical internet communities (eg. blockchains):

A typical resident of a geographic community spends most of their time and resources in that community, and so measured inequality in a geographic community reflects inequality in total resources available to people. **But in an internet community, measured inequality can come from two sources: (i) inequality in total resources available to different participants, and (ii) inequality in *level of interest* in participating in the community.**

The average person with $15 in fiat currency is poor and is missing out on the ability to have a good life. The average person with $15 in cryptocurrency is a dabbler who opened up a wallet once for fun. Inequality in level of interest is a healthy thing; every community has its dabblers and its full-time hardcore fans with no life. So if a cryptocurrency has a very high Gini coefficient, but it turns out that much of this inequality comes from inequality in level of interest, then the number points to a much less scary reality than the headlines imply.

Cryptocurrencies, even those that turn out to be highly plutocratic, will not turn any part of the world into anything close to dystopia A. But badly-distributed cryptocurrencies may well look like dystopia B, a problem compounded if [coin voting](#) governance [is used](#) to make protocol decisions. Hence, to detect the problems that cryptocurrency communities worry about most, we want a metric that captures proximity to dystopia B more specifically.

An alternative: measuring dystopia A problems and dystopia B problems separately

An alternative approach to measuring inequality involves directly estimating suffering from resources being unequally distributed (that is, "dystopia A" problems). First, start with some utility function representing the value of having a certain amount of money. $\log(x)$ is popular, because it captures the intuitively appealing approximation that doubling one's income is about as useful at any level: going from $10,000 to $20,000 adds the same utility as going from $5,000 to $10,000 or from $40,000 to $80,000). The score is then a matter of measuring how much utility is lost compared to if everyone just got the average income:

$$\log(\frac{\sum_{i=1}^n x_i}{n}) - \frac{\sum_{i=1}^n \log(x_i)}{n}$$

The first term (log-of-average) is the utility that everyone would have if money were perfectly redistributed, so everyone earned the average income. The second term (average-of-log) is the average utility in that economy today. The difference represents lost utility from inequality, if you look narrowly at resources as something used for personal consumption. There are other ways to define this formula, but they end up being close to equivalent (eg. the [1969 paper by Anthony Atkinson](#) suggested an "equally distributed equivalent level of income" metric which, in the $U(x) = \log(x)$ case, is just a monotonic function of the above, and the [Theil L index](#) is perfectly mathematically equivalent to the above formula).

To measure concentration (or "dystopia B" problems), the [Herfindahl-Hirschman index](#) is an excellent place to start, and is already used to measure economic concentration in industries:

$$\frac{\sum_{i=1}^n x_i^2}{(\sum_{i=1}^n x_i)^2}$$

Or for you visual learners out there:
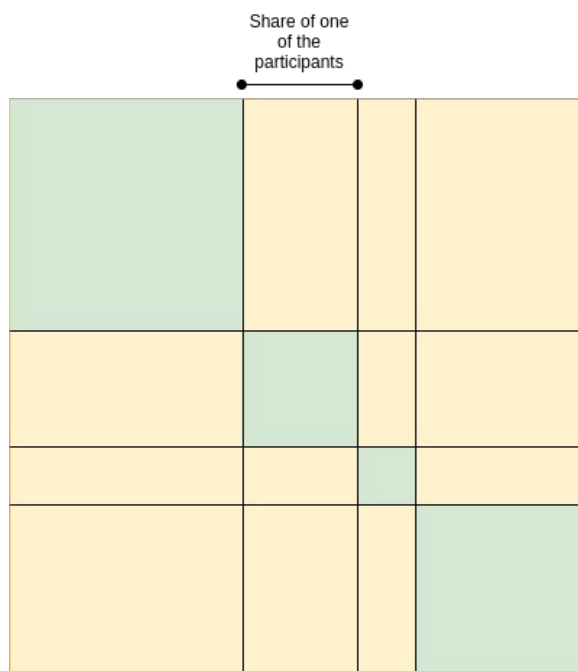
Herfindahl-Hirschman index: green area divided by total area.

There are other alternatives to this; the Theil T index has some similar properties though also some differences. A simpler-and-dumber alternative is the Nakamoto coefficient: the minimum number of participants needed to add up to more than 50% of the total. Note that all three of these concentration indices focus heavily on what happens near the top (and deliberately so): a large number of dabblers with a small quantity of resources contributes little or nothing to the index, while the act of two top participants merging can make a very big change to the index.

For cryptocurrency communities, where concentration of resources is one of the biggest risks to the system but where someone only having 0.00013 coins is not any kind of evidence that they're actually starving, adopting indices like this is the obvious approach. But even for countries, it's probably worth talking about, and measuring, concentration of power and suffering from lack of resources more separately.

That said, **at some point we have to move beyond even these indices**. The harms from concentration are not just a function of the size of the actors; they are also heavily dependent on the relationships between the actors and their ability to collude with each other. Similarly, resource allocation is network-dependent: lack of formal resources may not be that harmful if the person lacking resources has an informal network to tap into. But dealing with these issues is a much harder challenge, and so we do also need the simpler tools while we still have less data to work with.

Verkle trees

2021 Jun 18

*Special thanks to Dankrad Feist and Justin Drake for feedback and review.*

Verkle trees are shaping up to be an important part of Ethereum's upcoming scaling upgrades. They serve the same function as Merkle trees: you can put a large amount of data into a Verkle tree, and make a short proof ("witness") of any single piece, or set of pieces, of that data that can be verified by someone who only has the root of the tree. The key property that Verkle trees provide, however, is that they are *much more efficient* in proof size. If a tree contains a billion pieces of data, making a proof in a traditional binary Merkle tree would require about 1 kilobyte, but in a Verkle tree the proof would be *less than 150 bytes* - a reduction sufficient to make stateless clients finally viable in practice.

Verkle trees are still a new idea; they were first introduced by John Kuszmaul in this paper from 2018, and they are still not as widely known as many other important new cryptographic constructions. This post will explain what Verkle trees are and how the cryptographic magic behind them works. The price of their short proof size is a higher level of dependence on more complicated cryptography. That said, the cryptography still much simpler, in my opinion, than the advanced cryptography found in modern ZK SNARK schemes. In this post I'll do the best job that I can at explaining it.

Merkle Patricia vs Verkle Tree node structure

In terms of the *structure* of the tree (how the nodes in the tree are arranged and what they contain), a Verkle tree is very similar to the Merkle Patricia tree currently used in Ethereum. Every node is either (i) empty, (ii) a leaf node containing a key and value, or (iii) an intermediate node that has some fixed number of children (the "width" of the tree). The value of an intermediate node is computed as a hash of the values of its children.

The location of a value in the tree is based on its key: in the diagram below, to get to the node with key `4cc`, you start at the root, then go down to the child at position `4`, then go down to the child at position `c` (remember: `c = 12` in hexadecimal), and then go down again to the child at position `c`. To get to the node with key `baaa`, you go to the position-`b` child of the root, and then the position-`a` child of *that* node. The node at path `(b,a)` directly contains the node with key `baaa`, because there are no other keys in the tree starting with `ba`.
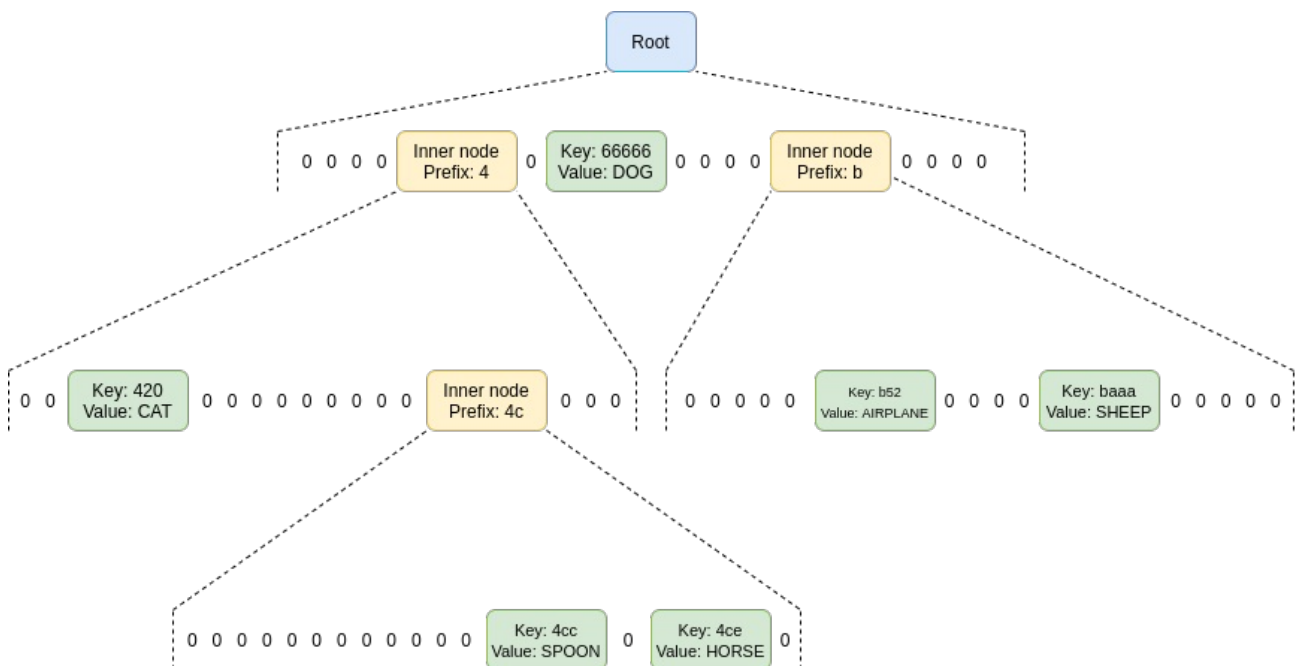

The structure of nodes in a hexary (16 children per parent) Verkle tree, here filled with six (key, value) pairs.

The only real difference in the *structure* of Verkle trees and Merkle Patricia trees is that Verkle trees are wider in practice. *Much* wider. Patricia trees are at their most efficient when `width = 2` (so Ethereum's hexary Patricia tree is actually quite suboptimal). Verkle trees, on the other hand, get shorter and shorter proofs the higher the width; the only limit is that if width gets *too* high, proofs start to take too long to create. The [Verkle tree proposed for Ethereum](#) has a width of 256, and some even favor raising it to 1024 (!!).

Commitments and proofs

In a Merkle tree (including Merkle Patricia trees), the proof of a value consists of the entire set of *sister nodes*: the proof must contain all nodes in the tree that *share a parent* with any of the nodes in the path going down to the node you are trying to prove. That may be a little complicated to understand, so here's a picture of a proof for the value in the `4ce` position. Sister nodes that must be included in the proof are highlighted in red.
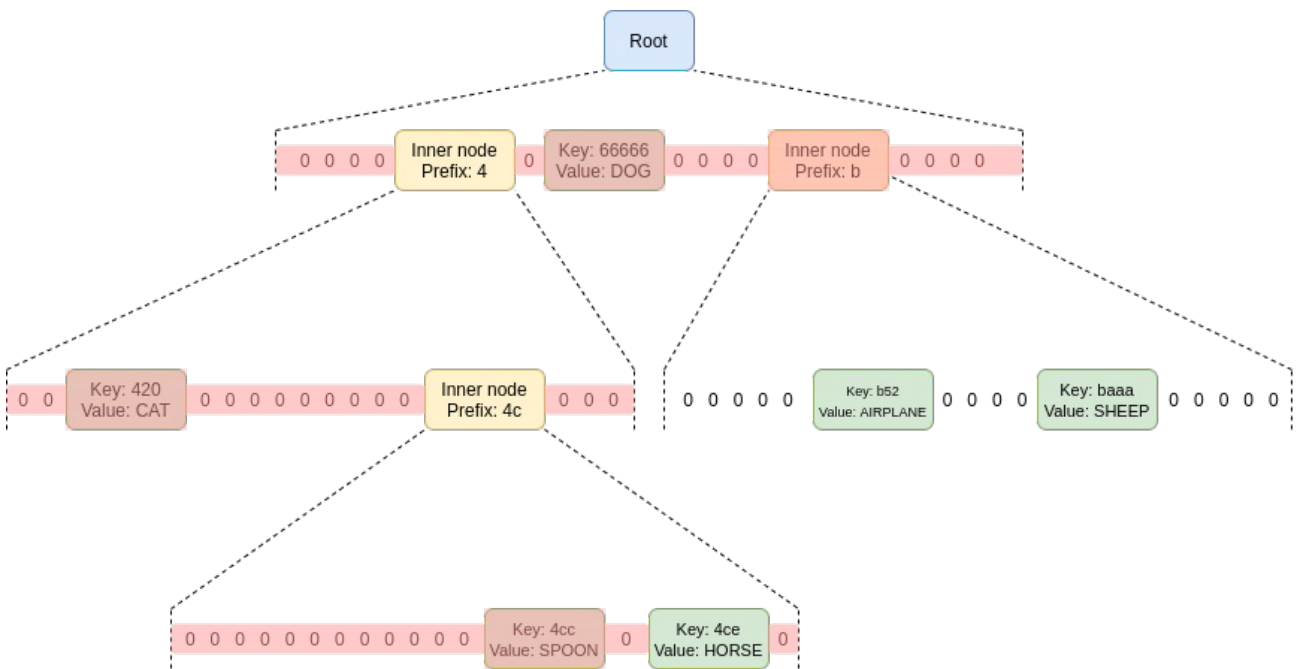

That's a lot of nodes! You need to provide the sister nodes at each level, because you need the entire set of children of a node to compute the value of that node, and you need to keep doing this until you get to the root. You might think that this is not that bad because most of the nodes are zeroes, but that's only because this tree has very few nodes. If this tree had 256 randomly-allocated nodes, the top layer would almost certainly have all 16 nodes full, and the second layer would on average be ~63.3% full.

In a Verkle tree, on the other hand, you do not need to provide sister nodes; instead, you just provide the path, with a little bit extra as a proof. This is why Verkle trees benefit from greater width and Merkle Patricia trees do not: a tree with greater width leads to shorter paths in both cases, but in a Merkle Patricia tree this effect is overwhelmed by the higher cost of needing to provide all the `width - 1` sister nodes per level in a proof. In a Verkle tree, that cost does not exist.

So what is this little extra that we need as a proof? To understand that, we first need to circle back to one key detail: the hash function used to compute an inner node from its children is not a regular hash. Instead, it's a **vector commitment**.

A vector commitment scheme is a special type of hash function, hashing a list $h(z_1, z_2 \ldots z_n) \rightarrow C$. But vector commitments have the special property that for a commitment C and a value z_i, it's possible to make a short proof that C is the commitment to some list where the value at the i'th position is z_i. In a Verkle proof, this short proof replaces the function of the sister nodes in a Merkle Patricia proof, giving the verifier confidence that a child node really is the child at the given

position of its parent node.
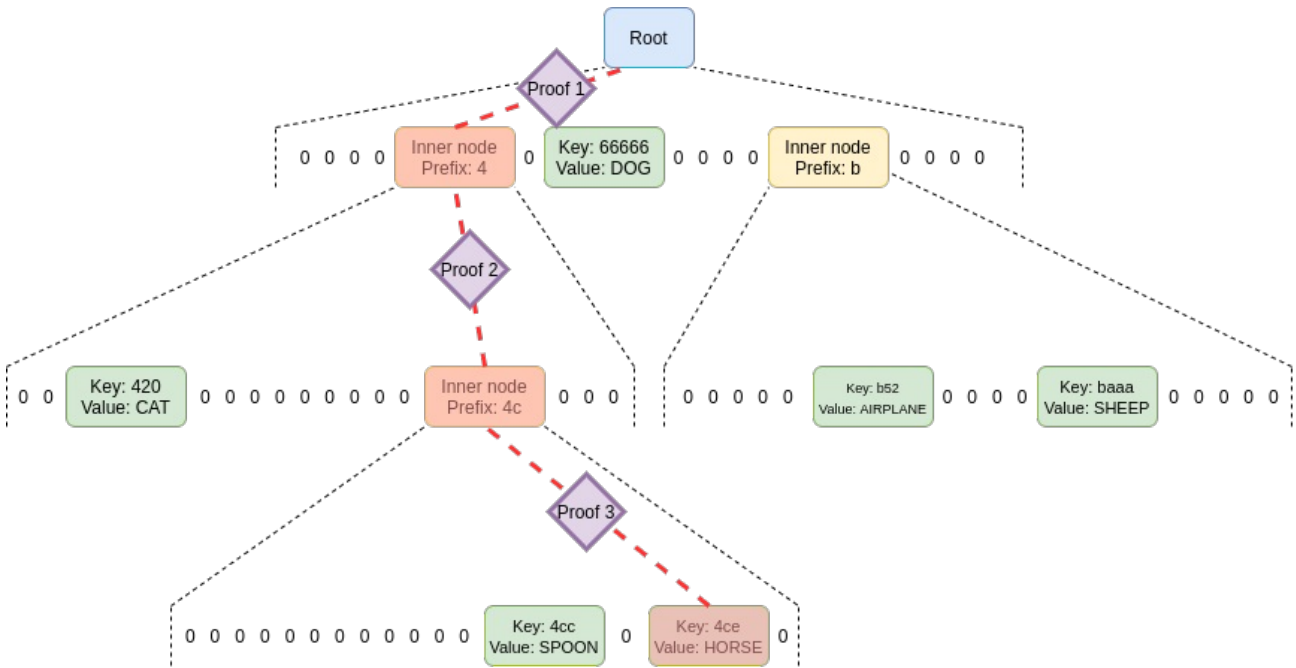
No sister nodes required in a proof of a value in the tree; just the path itself plus a few short proofs to link each commitment in the path to the next.

In practice, we use a primitive even more powerful than a vector commitment, called a **polynomial commitment**. Polynomial commitments let you hash a polynomial, and make a proof for the evaluation of the hashed polynomial at *any* point. You can use polynomial commitments as vector commitments: if we agree on a set of standardized coordinates $(c\_1, c\_2 ... c\_n)$, given a list $(y\_1, y\_2 ... y\_n)$ you can commit to the polynomial P where $P(c\_i) = y\_i$ for all $i \in [1..n]$ (you can find this polynomial with Lagrange interpolation). I talk about polynomial commitments at length in my article on ZK-SNARKs. The two polynomial commitment schemes that are the easiest to use are KZG commitments and bulletproof-style commitments (in both cases, a commitment is a single 32-48 byte elliptic curve point). Polynomial commitments give us more flexibility that lets us improve efficiency, and it just so happens that the simplest and most efficient vector commitments available *are* the polynomial commitments.

This scheme is already very powerful as it is: **if you use a KZG commitment and proof, the proof size is 96 bytes per intermediate node, nearly 3x more space-efficient than a simple Merkle proof** if we set width = 256. However, it turns out that we can increase space-efficiency even further.

Merging the proofs

Instead of requiring one proof for each commitment along the path, **by using the extra properties of polynomial commitments we can make a single fixed-size proof that proves *all* parent-child links between commitments along the paths for an unlimited number of keys**. We do this using a [scheme that implements multiproofs through random evaluation](#).

But to use this scheme, we first need to convert the problem into a more structured one. We have a proof of one or more values in a Verkle tree. The main part of this proof consists of the intermediary nodes along the path to each node. For each node that we provide, we also have to prove that it actually is the child of the node above it (and in the correct position). In our single-value-proof example above, we needed proofs to prove:

- That the `key: 4ce` node actually is the position-e child of the `prefix: 4c` intermediate node.
- That the `prefix: 4c` intermediate node actually is the position-c child of the `prefix: 4` intermediate node.
- That the `prefix: 4` intermediate node actually is the position-4 child of the root

If we had a proof proving multiple values (eg. both `4ce` and `420`), we would have even more nodes and even more linkages. But in any case, **what we are proving is a sequence of statements of the form "node A actually is the position-i child of node B"**. If we are using polynomial commitments, this turns into equations: $A(x_i) = y$, where y is the hash of the commitment to B.

The details of this proof are technical and better [explained by Dankrad Feist](#) than myself. By far the bulkiest and time-consuming step in the proof generation involves computing a polynomial g of the form:

$$g(X) = r^0\frac{A_0(X) - y_0}{X - x_0} + r^1\frac{A_1(X) - y_1}{X - x_1} + ... + r^n\frac{A_n(X) - y_n}{X - x_n}$$

It is only possible to compute each term $r^i\frac{A_i(X) - y_i}{X - x_i}$ if that expression is a polynomial (and not a fraction). And that requires $A_i(X)$ to equal y_i at the point x_i.

We can see this with an example. Suppose:

- $A_i(X) = X^2 + X + 3$
- We are proving for $(x_i = 2, y_i = 9)$. $A_i(2)$ does equal 9 so this will work.

$A_i(X) - 9 = X^2 + X - 6$, and $\frac{X^2 + X - 6}{X - 2}$ gives a clean $(X - 3)$. But if we tried to fit in $(x_i = 2, y_i = 10)$, this would not work; $X^2 + X - 7$ *cannot* be cleanly divided by $(X - 2)$

without a fractional remainder.

The rest of the proof involves providing a polynomial commitment to $g(X)$ and then proving that the commitment is actually correct. Once again, see [Dankrad's more technical description](#) for the rest of the proof.
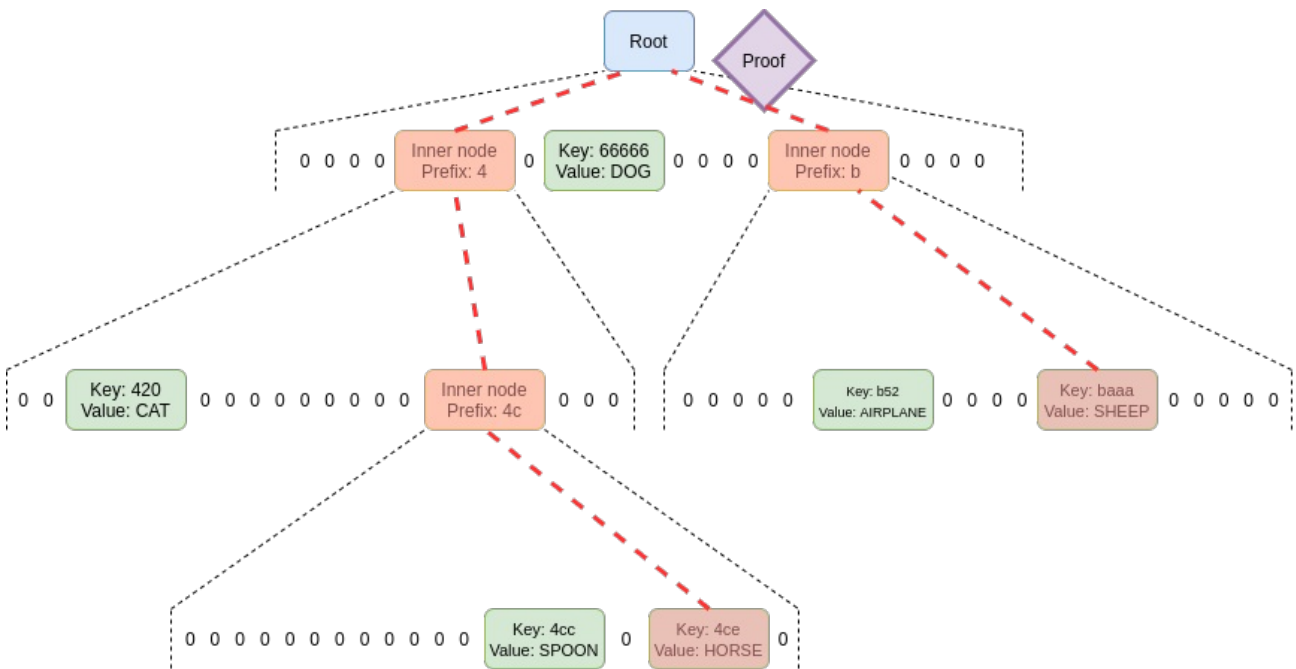
One single proof proves an unlimited number of parent-child relationships.

And there we have it, that's what a maximally efficient Verkle proof looks like.

Key properties of proof sizes using this scheme

- Dankrad's multi-random-evaluation proof allows the prover to **prove an arbitrary number of evaluations $A_i(x_i) = y_i$**, given commitments to each A_i and the values that are being proven. **This proof is constant size** (one polynomial commitment, one number, and two proofs; 128-1000 bytes depending on what scheme is being used).
- **The y_i values do not need to be provided explicitly**, as they can be directly computed from the other values in the Verkle proof: each y_i is itself the hash of the next value in the path (either a commitment or a leaf).
- **The x_i values also do not need to be provided explicitly**, since the paths (and hence the x_i values) can be computed from the keys and the coordinates derived from the paths.
- Hence, **all we need is the leaves (keys and values) that we are proving, as well as the commitments along the path from each leaf to the root**.
- Assuming a width-256 tree, and 2^{32} nodes, a proof would require the keys and values that are being proven, plus (on average) **three commitments for each value** along the path from that value to the root.
- **If we are proving many values, there are further savings**: no matter how many values you are proving, you will not need to provide more than the 256 values at the top level.

Proof sizes (bytes). Rows: tree size, cols: key/value pairs proven

	1	10	100	1,000	10,000
256	176	176	176	176	176
65,536	224	608	4,112	12,176	12,464
16,777,216	272	1,040	8,864	59,792	457,616
4,294,967,296	320	1,472	13,616	107,744	937,472

Assuming width 256, and 48-byte KZG commitments/proofs. Note also that this assumes a maximally even tree; for a realistic randomized tree, add a depth of ~0.6 (so ~30 bytes per element). If bulletproof-

style commitments are used instead of KZG, it's safe to go down to 32 bytes, so these sizes can be reduced by 1/3.

Prover and verifier computation load

The bulk of the **cost of generating a proof** is computing each $r^i\frac{A_i(X) - y_i}{X - x_i}$ expression. This requires roughly four field operations (ie. 256 bit modular arithmetic operations) times the width of the tree. This is the main constraint limiting Verkle tree widths. Fortunately, four field operations is a small cost: a single elliptic curve multiplication typically takes hundreds of field operations. Hence, Verkle tree widths can go quite high; width 256-1024 seems like an optimal range.

To _edit_ the tree, we need to "walk up the tree" from the leaf to the root, changing the intermediate commitment at each step to reflect the change that happened lower down. Fortunately, we don't have to re-compute each commitment from scratch. Instead, we take advantage of the homomorphic property: given a polynomial commitment $C = com(F)$, we can compute $C' = com(F + G)$ by taking $C' = C + com(G)$. In our case, $G = L_i * (v_{new} - v_{old})$, where L_i is a pre-computed commitment for the polynomial that equals 1 at the position we're trying to change and 0 everywhere else.

Hence, a single edit requires ~4 elliptic curve multiplications (one per commitment between the leaf and the root, this time including the root), though these can be sped up considerably by pre-computing and storing _many multiples_ of each L_i.

Proof verification is quite efficient. For a proof of N values, the verifier needs to do the following steps, all of which can be done within a hundred milliseconds for even thousands of values:

- One size-N [elliptic curve fast linear combination](#)
- About $4N$ field operations (ie. 256 bit modular arithmetic operations)
- A small constant amount of work that does not depend on the size of the proof

Note also that, like Merkle Patricia proofs, a Verkle proof gives the verifier enough information to _modify_ the values in the tree that are being proven and compute the new root hash after the changes are applied. This is critical for verifying that eg. state changes in a block were processed correctly.

Conclusions

Verkle trees are a powerful upgrade to Merkle proofs that allow for much smaller proof sizes. Instead of needing to provide all "sister nodes" at each level, the prover need only provide a single proof that proves _all_ parent-child relationships between all commitments along the paths from each leaf node to the root. This allows proof sizes to decrease by a factor of ~6-8 compared to ideal Merkle trees, and by a factor of over 20-30 compared to the hexary Patricia trees that Ethereum uses today (!!).

They do require more complex cryptography to implement, but they present the opportunity for large gains to scalability. In the medium term, SNARKs can improve things further: we can either SNARK the already-efficient Verkle proof verifier to reduce witness size to near-zero, or switch back to SNARKed Merkle proofs if/when SNARKs get much better (eg. [through GKR](#), or very-SNARK-friendly hash functions, or ASICs). Further down the line, the rise of quantum computing will force a change to STARKed Merkle proofs with hashes as it makes the linear homomorphisms that Verkle trees depend on insecure. But for now, they give us the same scaling gains that we would get with such more advanced technologies, and we already have all the tools that we need to implement them efficiently.

Blockchain voting is overrated among uninformed people but underrated among informed people

2021 May 25

Special thanks to Karl Floersch, Albert Ni, Mr Silly and others for feedback and discussion

Voting is a procedure that has a very important need for process integrity. The result of the vote must be correct, and this must be guaranteed by a transparent process so that everyone can be convinced that the result is correct. It should not be possible to successfully interfere with anyone's attempt to vote or prevent their vote from being counted.

Blockchains are a technology which is all about providing guarantees about process integrity. If a process is run on a blockchain, the process is guaranteed to run according to some pre-agreed code and provide the correct output. No one can prevent the execution, no one can tamper with the execution, and no one can censor and block any users' inputs from being processed.

So at first glance, it seems that blockchains provide exactly what voting needs. And I'm far from the only person to have had that thought; plenty of major prospective users are interested. But as it turns out, some people have a very different opinion....

FEATURE
Why blockchain-based voting could threaten democracy

As the desire to increase voter turnout remains strong and the number of online voting pilot projects rises in the U.S. and abroad, some security experts warn any internet-based election system is wide open to attack, regardless of the underlying infrastructure.

By Lucas Mearian
Senior Reporter, Computerworld | AUG 12, 2019 3:00 AM PDT

Despite the seeming perfect match between the needs of voting and the technological benefits that blockchains provide, we regularly see scary articles arguing against the combination of the two. And it's not just a single article: here's an anti-blockchain-voting piece from Scientific American, here's another from CNet, and here's another from ArsTechnica. And it's not just random tech journalists: Bruce Schneier is against blockchain voting, and researchers at MIT wrote a whole paper arguing that it's a bad idea. So what's going on?

Outline

There are **two key lines of criticism** that are most commonly levied by critics of blockchain voting protocols:

1. **Blockchains are the wrong software tool** to run an election. The trust properties they provide are not a good match for the properties that voting needs, and other kinds of software tools with different information flow and trust properties would work better.
2. **Software in general cannot be trusted** to run elections, no matter what software it is. The risk of undetectable software and hardware bugs is too high, no matter how the platform is organized.

This article will discuss both of these claims in turn ("refute" is too strong a word, but I definitely disagree more than I agree with both claims). First, I will discuss the security issues with existing attempts to use blockchains for voting, and how **the correct solution is not to abandon blockchains, but to combine them with other cryptographic technologies**. Second, I will address the concern about whether or not software (and hardware) can be trusted. The answer: **computer security is actually getting quite a bit better**, and we can work hard to continue that trend.

Over the long term, **insisting on paper permanently would be a *huge* handicap to our ability to make voting better**. One vote per N years is a 250-year-old form of democracy, and we can have much better democracy if voting were much more convenient and simpler, so that we could do it much more often.

Needless to say, this entire post is predicated on good blockchain scaling technology (eg. [sharding](#)) being available. Of course, if blockchains cannot scale, none of this can happen. But so far, development of this technology is proceeding quickly, and there's no reason to believe that it can't happen.

Bad blockchain voting protocols

Blockchain voting protocols get hacked all the time. Two years ago, a blockchain voting tech company called Voatz was all the rage, and many people were very excited about it. But last year, some MIT researchers discovered a [string of critical security vulnerabilities](#) in their platform. Meanwhile, in Moscow, a blockchain voting system that was going to be used for an upcoming election [was hacked](#), fortunately a month before the election took place.

The hacks were pretty serious. Here is a table of the attack capabilities that [researchers analyzing Voatz](#) managed to uncover:

Adversary	Attacker Capability				
	Suppress Ballot	Learn Secret Vote	Alter Ballot	Learn User's Identity	Learn User IP
Passive Network (§5.3)		✓			✓
Active Network (§5.3)	✓	✓			✓
3rd-Party ID Svc. (§5.4)	✓			✓	✓
Root On-Device (§5.1)	✓	✓	✓	✓	✓
Voatz API Server (§5.2)	✓	✓	✓	✓	✓

Table 1: Summary of Potential Attacks by Adversary Type: Here we show what kind of adversary is capable of executing what sort of attack; e.g. a Passive Network adversary is capable of learning a user's secret ballot, and the User's IP. The viability of some of these attacks are dependent on the configuration of the particular election, (the ballot style, metadata, etc.), see the relevant section listed for explicit details.

This by itself is not an argument against *ever* using blockchain voting. But it is an argument that blockchain voting software should be designed more carefully, and scaled up slowly and incrementally over time.

Privacy and coercion resistance

But even the blockchain voting protocols that are not technically broken often suck. To understand why, we need to delve deeper into *what specific security properties* blockchains provide, and what specific security properties voting needs - when we do, we'll see that there is a mismatch.

Blockchains provide two key properties: **correct execution** and **censorship resistance**. Correct execution just means that the blockchain accepts inputs ("transactions") from users, correctly processes them according to some pre-defined rules, and returns the correct output (or adjusts the

blockchain's "state" in the correct way). Censorship resistance is also simple to understand: any user that *wants* to send a transaction, and is willing to pay a high enough fee, *can* send the transaction and expect to see it quickly included on-chain.

Both of these properties are very important for voting: you want the output of the vote to actually be the result of counting up the number of votes for each candidate and selecting the candidate with the most votes, and you definitely want anyone who is eligible to vote to be able to vote, even if some powerful actor is trying to block them. **But voting also requires some crucial properties that blockchains *do not* provide**:

- **Privacy**: you should not be able to tell which candidate someone specific voted for, or even if they voted at all
- **Coercion resistance**: you should not be able to *prove* to someone else how you voted, *even if you want to*

The need for the first requirement is obvious: you want people to vote based on their personal feelings, and not how people around them or their employer or the police or random thugs on the street will feel about their choice. The second requirement is needed to prevent vote selling: if you can prove how you voted, selling your vote becomes very easy. Provability of votes would also enable forms of coercion where the coercer demands to see some kind of proof of voting for their preferred candidate. Most people, even those aware of the first requirement, do not think about the second requirement. But the second requirement is also necessary, and it's quite technically nontrivial to provide it. **Needless to say, the average "blockchain voting system" that you see in the wild does not even try to provide the second property, and usually fails at providing the first**.

Secure electronic voting without blockchains

The concept of cryptographically secured execution of social mechanisms was not invented by blockchain geeks, and indeed existed far before us. Outside the blockchain space, there is a 20-year-old tradition of cryptographers working on the secure electronic voting problem, and the good news is that there *have* been solutions. An important paper that is cited by much of the literature of the last two decades is Juels, Catalano and Jakobsson's 2002 paper titled "Coercion-Resistant Electronic Elections":

Coercion-Resistant Electronic Elections

Ari Juels[1] and Dario Catalano[2] and Markus Jakobsson[3]

[1] RSA Laboratories
Bedford, MA, USA
e-mail: ajuels@rsasecurity.com

[2] CNRS-Ecole Normale Supérieure
75230 Paris Cedex 05 - France, France
e-mail: dario.catalano@ens.fr

[3] Indiana University, School of Informatics
Bloomington, IN, USA
e-mail: markus@indiana.edu

Abstract. We introduce a model for electronic election schemes that involves a more powerful adversary than in previous work. In particular, we allow the adversary to demand of coerced voters that they vote in a particular manner, abstain from voting, or even disclose their secret keys. We define a scheme to be *coercion-resistant* if it is infeasible for the adversary to determine whether a coerced voter complies with the demands.

Since then, there have been many iterations on the concept; Civitas is one prominent example, though there are also many others. These protocols all use a similar set of core techniques. There is an agreed-upon set of "talliers" and there is a trust assumption that the majority of the talliers is honest. The talliers each have "shares" of a private key secret-shared among themselves, and the corresponding public key is published. Voters publish votes encrypted to the talliers' public key, and talliers use a secure multi-party computation (MPC) protocol to decrypt and verify the votes and compute the tally. The tallying computation is done "inside the MPC": the talliers never learn their private key, and they compute the final result without learning anything about any individual vote beyond what can be learned from looking at the final result itself.

Encrypting votes provides privacy, and some additional infrastructure such as [mix-nets](#) is added on top to make the privacy stronger. To provide coercion resistance, one of two techniques is used. One option is that during the registration phase (the phase in which the talliers learn each registered voter's public key), the voter generates or receives a secret key. The corresponding public key is secret shared among the talliers, and the talliers' MPC only counts a vote if it is signed with the secret key. A voter has no way to prove to a third party what their secret key is, so if they are bribed or coerced they can simply show and cast a vote signed with the wrong secret key. Alternatively, a voter could have the ability to send a message to *change* their secret key. A voter has no way of proving to a third party that they did *not* send such a message, leading to the same result.

The second option is a technique where voters can make multiple votes where the second overrides the first. If a voter is bribed or coerced, they can make a vote for the briber/coercer's preferred candidate, but later send another vote to override the first.
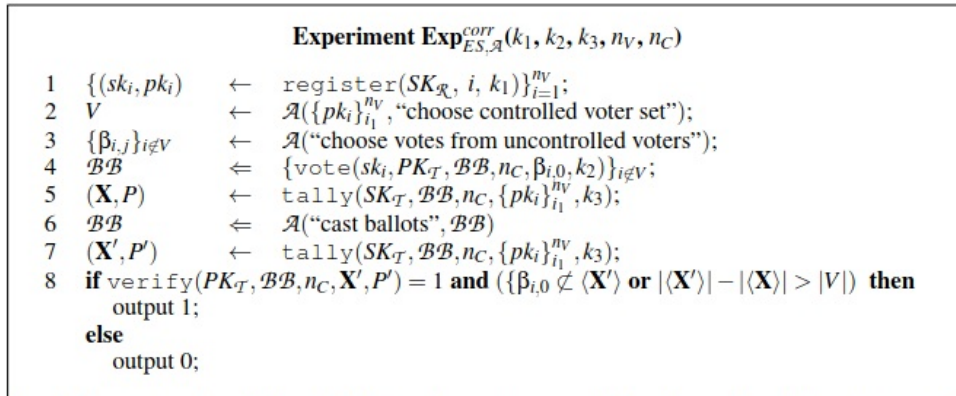

Experiment $\text{Exp}_{ES,\mathcal{A}}^{corr}(k_1, k_2, k_3, n_V, n_C)$

$$
\begin{array}{lll}
1 & \{(sk_i, pk_i)\} & \leftarrow \texttt{register}(SK_\mathcal{R}, i, k_1)\}_{i=1}^{n_V}; \\
2 & V & \leftarrow \mathcal{A}(\{pk_i\}_{i_1}^{n_V}, \text{``choose controlled voter set''}); \\
3 & \{\beta_{i,j}\}_{i \notin V} & \leftarrow \mathcal{A}(\text{``choose votes from uncontrolled voters''}); \\
4 & \mathcal{BB} & \Leftarrow \{\texttt{vote}(sk_i, PK_\mathcal{T}, \mathcal{BB}, n_C, \beta_{i,0}, k_2)\}_{i \notin V}; \\
5 & (\mathbf{X}, P) & \leftarrow \texttt{tally}(SK_\mathcal{T}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3); \\
6 & \mathcal{BB} & \Leftarrow \mathcal{A}(\text{``cast ballots''}, \mathcal{BB}) \\
7 & (\mathbf{X}', P') & \leftarrow \texttt{tally}(SK_\mathcal{T}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3); \\
8 & \textbf{if } \texttt{verify}(PK_\mathcal{T}, \mathcal{BB}, n_C, \mathbf{X}', P') = 1 \textbf{ and } (\{\beta_{i,0} \not\subset \langle\mathbf{X}'\rangle \textbf{ or } |\langle\mathbf{X}'\rangle| - |\langle\mathbf{X}\rangle| > |V|) \textbf{ then}
\end{array}
$$

   output 1;

else

   output 0;

Fig. 2: Experiment $\mathbf{Exp}_{ES,\mathcal{A}}^{corr}$. A tallying scheme is correct if the last, and only the last, vote of legitimate voters is counted.

Giving voters the ability to make a later vote that can override an earlier vote is the key coercion-resistance mechanism of [this protocol from 2015.](#)

Now, we get to a key important nuance in all of these protocols. They all rely on an outside primitive to complete their security guarantees: the **bulletin board** (this is the "BB" in the figure above). The bulletin board is a place where any voter can send a message, with a guarantee that (i) anyone can read the bulletin board, and (ii) anyone can send a message to the bulletin board that gets accepted. Most of the coercion-resistant voting papers that you can find will casually reference the existence of a bulletin board ([eg.](#) "as is common for electronic voting schemes, we assume a publicly accessible append-only bulletin board"), but far fewer papers talk about how this bulletin board can actually be *implemented*. And here, you can hopefully see where I am going with this: **the most secure way to implement a bulletin board *is* to just use an existing blockchain!**

Secure electronic voting with blockchains

Of course, there have been plenty of pre-blockchain attempts at making a bulletin board. [This paper from 2008](#) is such an attempt; its trust model is a standard requirement that "k of `n` servers must be honest" (`k = n/2` is common). [This literature review from 2021](#) covers some pre-blockchain attempts at bulletin boards as well as exploring the use of blockchains for the job; the pre-blockchain solutions reviewed similarly rely on a k-of-n trust model.

A blockchain is also a k-of-n trust model; it requires at least half of miners or proof of stake validators to be following the protocol, and if that assumption fails that often results in a "51% attack". So why is a blockchain better than a special purpose bulletin board? The answer is: setting up a k-of-n system that's actually trusted is hard, and blockchains are the only system that has already solved it, and at scale. Suppose that some government announced that it was making a voting system, and provided a list of 15 local organizations and universities that would be running a special-purpose bulletin board. How would you, as an outside observer, know that the government didn't just choose those 15 organizations from a list of 1000 based on their willingness to secretly collude with an intelligence agency?
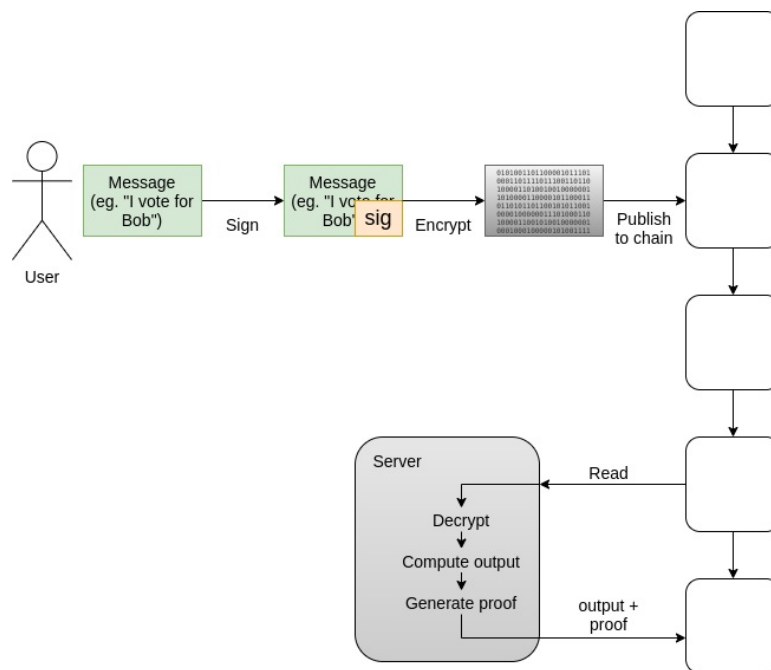
Public blockchains, on the other hand, have permissionless economic consensus

mechanisms (proof of work or proof of stake) that anyone can participate in, and they have an existing diverse and highly incentivized infrastructure of block explorers, exchanges and other watching nodes to constantly verify in real time that nothing bad is going on.

These more sophisticated voting systems are not *just* using blockchains; they rely on cryptography such as [zero knowledge proofs](#) to guarantee correctness, and on multi-party computation to guarantee coercion resistance. Hence, they avoid the weaknesses of more naive systems that simply just "put votes directly on the blockchain" and ignore the resulting privacy and coercion resistance issues. However, the blockchain bulletin board is nevertheless a key part of the security model of the whole design: if the committee is broken but the blockchain is not, coercion resistance is lost but all the other guarantees around the voting process still remain.

MACI: coercion-resistant blockchain voting in Ethereum

The Ethereum ecosystem is currently experimenting with a [system called MACI](#) that combines together a blockchain, ZK-SNARKs and a single central actor that guarantees coercion resistance (but has no power to compromise any properties other than coercion resistance). MACI is not very technically difficult. Users participate by signing a message with their private key, encrypting the signed message to a public key published by a central server, and publishing the encrypted signed message to the blockchain. The server downloads the messages from the blockchain, decrypts them, processes them, and outputs the result along with a ZK-SNARK to ensure that they did the computation correctly.

Users cannot prove how they participated, because they have the ability to send a "key change" message to trick anyone trying to audit them: they can first send a key change message to change their key from A to B, and then send a "fake message" signed with A. The server would reject the message, but no one else would have any way of knowing that the key change message had ever been sent. There is a trust requirement on the server, though only for privacy and coercion resistance; the server cannot publish an incorrect result either by computing incorrectly or by censoring messages. In the long term, multi-party computation can be used to decentralize the server somewhat, strengthening the privacy and coercion resistance guarantees.

There is a working demo of this scheme at [clr.fund](#) being used for quadratic funding. The use of the Ethereum blockchain to ensure censorship resistance of votes ensures a much higher degree of censorship resistance than would be possible if a committee was relied on for this instead.

Recap

- The voting process has four important security requirements that must be met for a vote to be

secure: **correctness**, **censorship resistance**, **privacy** and **coercion resistance**.
- Blockchains are good at the first two. They are bad at the last two.
- **Encryption** of votes put on a blockchain can add privacy. **Zero knowledge proofs** can bring back correctness despite observers being unable to add up votes directly because they are encrypted.
- **Multi-party computation** decrypting and checking votes can provide coercion resistance, if combined with a mechanic where users can interact with the system multiple times; either the first interaction invalidates the second, or vice versa
- Using a blockchain ensures that you have very high-security censorship resistance, and you keep this censorship resistance *even if* the committee colludes and breaks coercion resistance. **Introducing a blockchain can significantly increase the level of security of the system.**

But can technology be trusted?

But now we get back to the second, deeper, critique of electronic voting of any kind, blockchain or not: that technology itself is too insecure to be trusted.

The recent MIT paper criticizing blockchain voting includes this helpful table, depicting *any* form of paperless voting as being fundamentally too difficult to secure:

Table 1: **Four categories of voting systems.** The top row (green) is *software-independent* and far less vulnerable to serious failure than the bottom row (red). The bottom row is highly vulnerable and thus unsuitable for use in political elections, as explained further in §2.

	In person	Remote
Voter-verifiable paper ballots[3]	*Precinct voting*	*Mail-in ballots*
Unverifiable or electronic ballots	*DRE[4] voting machines*	*Internet/mobile/blockchain voting*

The key property that the authors focus on is **software-independence**, which they define as "the property that an undetected change or error in a system's software cannot cause an undetectable change in the election outcome". Basically, a bug in the code should not be able to accidentally make Prezzy McPresidentface the new president of the country (or, more realistically, a deliberately inserted bug should not be able to increase some candidate's share from 42% to 52%).

But there are other ways to deal with bugs. For example, any blockchain-based voting system that uses publicly verifiable zero-knowledge proofs can be independently verified. Someone can write their own implementation of the proof verifier and verify the Zk-SNARK themselves. They could even write their own software to vote. Of course, the technical complexity of actually doing this is beyond 99.99% of any realistic voter base, but if thousands of independent experts have the ability to do this and verify that it works, that is more than good enough in practice.

To the MIT authors, however, that is not enough:

> Thus, any system that is electronic only, even if end-to-end verifiable, seems unsuitable for political elections in the foreseeable future. The U.S. Vote Foundation has noted the promise of E2E-V methods for improving online voting security, but has issued a detailed report recommending avoiding their use for online voting unless and until the technology is far more mature and fully tested in pollsite voting [38].

> Others have proposed extensions of these ideas. For example, the proposal of Juels et al. [55] emphasizes the use of cryptography to provide a number of forms of "coercion resistance." The Civitas proposal of Clarkson et al. [24] implements additional mechanisms for coercion resistance, which Iovino et al. [53] further incorporate and elaborate into their Selene system. From our perspective, these proposals are innovative but unrealistic: they are quite complex, and most seriously, their security relies upon voters' devices being uncompromised and functioning as intended, an unrealistic assumption.

The problem that the authors focus on is not the *voting system's hardware* being secure; risks on that side actually can be mitigated with zero knowledge proofs. Rather, the authors focus on a different security problem: can *users' devices* even in principle be made secure?

Given the long history of all kinds of exploits and hacks of consumer devices, one would be very justified in thinking the answer is "no". Quoting my own [article on Bitcoin wallet security from 2013](#):

> Last night around 9PM PDT, I clicked a link to go to CoinChat[.]freetzi[.]com – and I was prompted to run java. I did (thinking this was a legitimate chatoom), and nothing happened. I closed the window and thought nothing of it. I opened my bitcoin-qt wallet approx 14 minutes later, and saw a transaction that I did NOT approve go to wallet 1Es3QVvKN1qA2p6me7jLCVMZpQXVXWPNTC for almost my entire wallet...

And:

> In June 2011, the Bitcointalk member "allinvain" lost 25,000 BTC (worth $500,000 at the time) after an unknown intruder somehow gained direct access to his computer. The attacker was able to access allinvain's wallet.dat file, and quickly empty out the wallet – either by sending a transaction from allinvain's computer itself, or by simply uploading the wallet.dat file and emptying it on his own machine.

But these disasters obscure a greater truth: **over the past twenty years, computer security has actually been [slowly and steadily improving](#)**. Attacks are much harder to find, often requiring the attacker to find bugs in multiple sub-systems instead of finding a single hole in a large complex piece of code. High-profile incidents are larger than ever, but this is not a sign that anything is getting less secure; rather, it's simply a sign that we are becoming much more dependent on the internet.

[Trusted hardware](#) is a very important recent source of improvements. Some of the new "blockchain phones" (eg. [this one from HTC](#)) go quite far with this technology and put a minimalistic security-focused operating system on the trusted hardware chip, allowing high-security-demanding applications (eg. cryptocurrency wallets) to stay separate from the other applications. Samsung has started making phones using [similar](#) [technology](#). And even devices that are never advertised as "blockchain devices" (eg. iPhones) frequently have trusted hardware of some kind. Cryptocurrency hardware wallets are effectively the same thing, except the trusted hardware module is physically located outside the computer instead of inside it. Trusted hardware (deservedly!) often gets a bad rap in security circles and especially the blockchain community, because it just [keeps](#) getting [broken](#) again and [again](#). And indeed, you definitely don't want to use it to *replace* your security protection. But as an *augmentation*, it's a huge improvement.

Finally, single applications, like cryptocurrency wallets and voting systems, are much simpler and have less room for error than an entire consumer operating system - even if you have to incorporate support for [quadratic voting](#), [sortition](#), [quadratic sortition](#) and whatever horrors the next generation's Glen Weyl invents in 2040. The benefit of tools like trusted hardware is their ability to *isolate* the simple thing from the complex and possibly broken thing, and these tools are having some success.

So the risks might decrease over time. But what are the benefits?

These improvements in security technology point to a future where consumer hardware might be more trusted in the future than it is today. Investments made in this area in the last few years are likely to keep paying off over the next decade, and we could expect further significant improvements. But what are the benefits of making voting electronic (blockchain based or otherwise) that justify exploring this whole space?

My answer is simple: voting would become much more efficient, allowing us to do it much more often. Currently, formal democratic input into organizations (governmental *or* corporate) tends to be limited to a single vote once every 1-6 years. This effectively means that each voter is only putting less than one bit of input into the system each year. Perhaps in large part as a result of this, decentralized decision-making in our society is heavily bifurcated into two extremes: pure democracy and pure markets. Democracy is either very inefficient (corporate and government votes) or very insecure (social media likes/retweets). Markets are far more technologically efficient and are much more secure than social media, but their fundamental economic logic makes them a poor fit for many kinds of decision problems, particularly having to do with public goods.
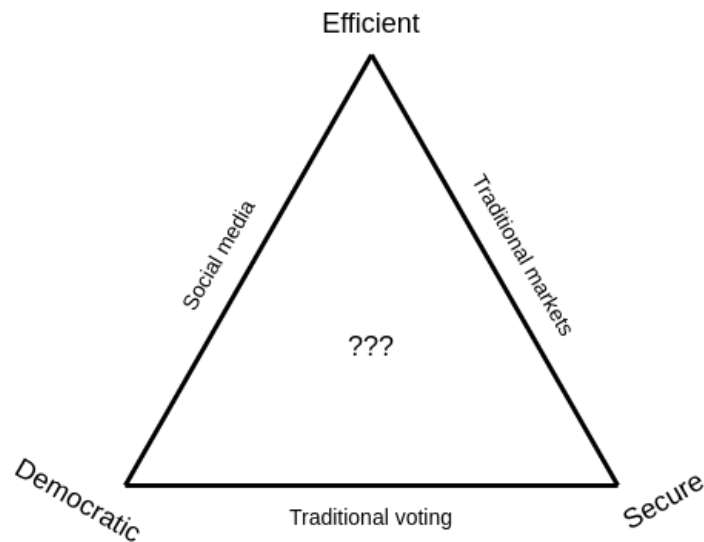
Yes, I know it's yet another triangle, and I really really apologize for having to use it. But please bear with me just this once.... (ok fine, I'm sure I'll make even more triangles in the future; just suck it up and deal with it)

There is a lot that we could do if we could build more systems that are somewhere in between democracy and markets, benefiting from the egalitarianism of the former, the technical efficiency of the latter and economic properties all along the spectrum in between the two extremes. Quadratic funding is an excellent example of this. Liquid democracy is another excellent example. Even if we don't introduce fancy new delegation mechanisms or quadratic math, there's a lot that we could do by doing voting *much more* and at smaller scales more adapted to the information available to each individual voter. But the challenge with all of these ideas is that in order to have a scheme that durably maintains *any* level of democraticness at all, you need some form of sybil resistance and vote-buying mitigation: exactly the problems that these fancy ZK-SNARK + MPC + blockchain voting schemes are trying to solve.

The crypto space can help

One of the underrated benefits of the crypto space is that it's an excellent "virtual special economic zone" for testing out economic and cryptographic ideas in a highly adversarial environment. Whatever you build and release, once the economic power that it controls gets above a certain size, a whole host of diverse, sometimes altruistic, sometimes profit-motivated, and sometimes malicious actors, many of whom are completely anonymous, will descend upon the system and try to twist that economic power toward their own various objectives.

The incentives for attackers are high: if an attacker steals $100 from your cryptoeconomic gadget, they can often get the full $100 in reward, and they can often get away with it. But the incentives for defenders are also high: if you develop a tool that helps users *not* lose their funds, you could (at least sometimes) turn that into a tool and earn millions. Crypto is the ultimate training zone: if you can build something that can survive in this environment at scale, it can probably also survive in the bigger world as well.

This applies to quadratic funding, it applies to multisig and social recovery wallets, and it can apply to voting systems too. The blockchain space has already helped to motivate the rise of important security technologies:

- Hardware wallets
- Efficient general-purpose zero knowledge proofs
- Formal verification tools
- "Blockchain phones" with trusted hardware chips
- Anti-sybil schemes like Proof of Humanity

In all of these cases, some version of the technology existed before blockchains came onto the scene. But it's hard to deny that blockchains have had a significant impact in pushing these efforts forward, and the large role of incentives inherent to the space plays a key role in raising the stakes enough for

the development of the tech to actually happen.

Conclusion

In the short term, any form of blockchain voting should certainly remain confined to small experiments, whether in small trials for more mainstream applications or for the blockchain space itself. Security is at present definitely not good enough to rely on computers for everything. But it's improving, and if I am wrong and security *fails* to improve then not only blockchain voting, but also cryptocurrency as a whole, will have a hard time being successful. Hence, there is a large incentive for the technology to continue to improve.

We should all continue watching the technology and the efforts being made everywhere to try and increase security, and slowly become more comfortable using technology in very important social processes. Technology is *already* key in our financial markets, and a crypto-ization of a large part of the economy (or even just replacing gold) will put an even greater portion of the economy into the hands of our cryptographic algorithms and the hardware that is running them. We should watch and support this process carefully, and over time take advantage of its benefits to bring our governance technologies into the 21st century.

The Limits to Blockchain Scalability

2021 May 23

Special thanks to Felix Lange, Martin Swende, Marius van der Wijden and Mark Tyneway for feedback and review.

Just how far can you push the scalability of a blockchain? Can you really, as Elon Musk wishes, "speed up block time 10X, increase block size 10X & drop fee 100X" without leading to extreme centralization and compromising the fundamental properties that make a blockchain what it is? If not, how far can you go? What if you change the consensus algorithm? Even more importantly, what if you change the technology to introduce features such as ZK-SNARKs or sharding? A sharded blockchain can theoretically just keep adding more shards; is there such a thing as adding too many?

As it turns out, there are important and quite subtle technical factors that limit blockchain scaling, both with sharding and without. In many cases there are solutions, but even with the solutions there are limits. This post will go through what many of these issues are.

Just increase the parameters, and all problems are solved. But at what cost?

It's crucial for blockchain decentralization for regular users to be able to run a node

At 2:35 AM, you receive an emergency call from your partner on the opposite side of the world who helps run your mining pool (or it could be a staking pool). Since about 14 minutes ago, your partner tells you, your pool and a few others split off from the chain which still carries 79% of the network. According to your node, the majority chain's blocks are invalid. There's a balance error: the key block appeared to erroneously assign 4.5 million extra coins to an unknown address.

An hour later, you're in a telegram chat with the other two small pools who were caught blindsided just as you were, as well as some block explorers and exchanges. You finally see someone paste a link to a tweet, containing a published message. "Announcing new on-chain sustainable protocol development fund", the tweet begins.

By the morning, arguments on Twitter, and on the one community forum that was not censoring the discussion, discussions are everywhere. But by then a significant part of the 4.5 million coins had been converted on-chain to other assets, and billions of dollars of defi transactions had taken place. 79% of the consensus nodes, and all the major block explorers and endpoints for light wallets, were following this new chain. Perhaps the new dev fund will fund some development, or perhaps it will just all be embezzled by the leading pools and exchanges and their cronies. But regardless of how it turns out, the fund is for all intents and purposes a fait accompli, and regular users have no way to fight back.

Movie coming soon. Maybe it can be funded by MolochDAO or something.

Can this happen on your blockchain? The elites of your blockchain community, including pools, block explorers and hosted nodes, are probably quite well-coordinated; quite likely they're all in the same telegram channels and wechat groups. If they really *want* to organize a sudden change to the protocol rules to further their own interests, then they probably can. The Ethereum blockchain has fully resolved consensus failures in ten hours; if your blockchain has only one client implementation, and you only need to deploy a code change to a few dozen nodes, coordinating a change to client code can be done much faster. The only reliable way to make this kind of coordinated social attack *not* effective is through passive defense from the one constituency that actually is decentralized: the *users*.

Imagine how the story would have played out if the users were running nodes that verify the chain (whether directly or through more advanced indirect techniques), and automatically reject blocks that break the protocol rules *even if* over 90% of the miners or stakers support those blocks. If *every* user ran a verifying node, then the attack would have quickly failed: a few mining pools and exchanges would have forked off and looked quite foolish in the process. But even if *some* users ran verifying nodes, the attack would not have led to a *clean victory* for the attacker; rather, it would have led to *chaos,* with different users seeing different views of the chain. At the very least, the ensuing market panic and likely persistent chain split would greatly reduce the attackers' profits. The thought of navigating such a protracted conflict would itself deter most attacks.

Listen to Hasu on this one.

If you have a community of 37 node runners and 80000 passive listeners that check signatures and block headers, the attacker wins. If you have a community where everyone runs a node, the attacker loses. We don't know what the exact threshold is at which herd immunity against coordinated attacks kicks in, but there is one thing that's absolutely clear: more nodes good, fewer nodes bad, and we definitely need more than a few dozen or few hundred.

So, what are the limits to how much work we can require full nodes to do?

To maximize the number of users who can run a node, we'll focus on regular consumer hardware. There are some increases to capacity that can be achieved by demanding some specialized hardware purchases that are easy to obtain (eg. from Amazon), but they actually don't increase scalability by that much.

There are three key limitations to a full node's ability to process a large number of transactions:

- **Computing power**: what % of the CPU can we safely demand to run a node?
- **Bandwidth**: given the realities of current internet connections, how many *bytes* can a block contain?
- **Storage**: how many gigabytes on disk can we require users to store? Also, how quickly must it be readable? (ie. is HDD okay or do we need SSD)

Many erroneous takes on how far a blockchain can scale using "simple" techniques stem from overly optimistic estimates for each of these numbers. We can go through these three factors one by one:

Computing power

- **Bad answer**: 100% of CPU power can be spent on block verification
- **Correct answer**: ~5-10% of CPU power can be spent on block verification

There are four key reasons why the limit is so low:

- We need a safety margin to cover the possibility of DoS attacks (transactions crafted by an attacker to take advantage of weaknesses in code to take longer to process than regular transactions)
- Nodes need to be able to sync the chain after being offline. If I drop off the network for a minute, I should be able to catch up in a few seconds
- Running a node should not drain your battery very quickly and make all your other apps very slow
- There are other non-block-production tasks that nodes need to do as well, mostly around verifying and responding to incoming transactions and requests on the p2p network

Note that up until recently, most explanations for "why only 5-10%?" focused on a different problem: that because PoW blocks come at random times, it taking a long time to verify blocks increases the risk that multiple blocks get created at the same time. There are many fixes to this problem (eg. Bitcoin NG, or just using proof of stake). But these fixes do NOT solve the other four problems, and so they don't enable large gains in scalability as many had initially thought.

Parallelism is also not a magic bullet. Often, even clients of seemingly single-threaded blockchains are parallelized already: signatures can be verified by one thread while execution is done by other threads, and there's a separate thread that's handling transaction pool logic in the background. And the closer you get to 100% usage across *all* threads, the more energy-draining running a node becomes and the lower your safety margin against DoS.

Bandwidth

- **Bad answer**: if we have 10 MB blocks every 2-3 seconds, then most users have a >10 MB/sec network, so of course they can handle it
- **Correct answer**: *maybe* we can handle 1-5 MB blocks every 12 seconds. It's hard though.

Nowadays we frequently hear very high advertised statistics for how much bandwidth internet connections can offer: numbers of 100 Mbps and even 1 Gbps are common to hear. However, there is a large difference between advertised bandwidth and the expected actual bandwidth of a connection for several reasons:

1. "Mbps" refers to "millions of *bits* per second"; a bit is 1/8 of a byte, so you need to divide advertised bit numbers by 8 to get the advertised byte numbers.
2. Internet providers, just like all companies, often lie.
3. There's always multiple applications using the same internet connection, so a node can't hog the entire bandwidth.
4. p2p networks inevitably introduce their own overhead: nodes often end up downloading and re-uploading the same block multiple times (not to mention transactions being broadcasted through the mempool *before* being included in a block).

When Starkware did an experiment in 2019 where they published 500 kB blocks after the transaction data gas cost decrease made that possible for the first time, a few nodes were actually unable to handle blocks of that size. Ability to handle large blocks has since been improved and will continue to be improved. But no matter what we do, we'll still be *very* far from being able to naively take the average bandwidth in MB/sec, convince ourselves that we're okay with 1s latency, and be able to have blocks that are that size.

Storage

- **Bad answer**: 10 terabytes
- **Correct answer**: 512 gigabytes

The main argument here is, as you might guess, the same as elsewhere: the difference between theory and practice. *In theory*, there are 8 TB solid state drives that you can buy on Amazon (you *do* need SSDs or NVME; HDDs are too slow for storing the blockchain state). *In practice*, the laptop that was used to write this blog post has 512 GB, and if you make people go buy their own hardware, many of them will just get lazy (or they can't afford $800 for an 8 TB SSD) and use a centralized provider. And even if you can fit a blockchain onto some storage, a high level of activity can easily quickly burn through the disk and force you to keep getting a new one.
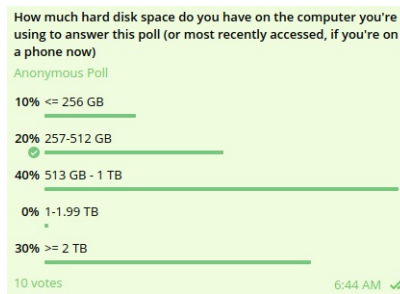
A poll in a group of blockchain protocol researchers of how much disk space everyone has. Small sample size, I know, but still...

Additionally, storage size determines the time needed for a new node to be able to come online and start participating in the network. Any data that existing nodes have to store is data that a new node has to download. This initial sync time (and bandwidth) is also a major barrier to users being able to run nodes. While writing this blog post, syncing a new geth node took me ~15 hours. If Ethereum had 10x more usage, syncing a new geth node would take at least a week, and it would be much more likely to just lead to your internet connection getting throttled. This is all even more important during an attack, when a successful response to the attack will likely involve many users spinning up new nodes when they were not running nodes before.

Interaction effects

Additionally, there are interaction effects between these three types of costs. Because databases use tree structures internally to store and retrieve data, the cost of fetching data from a database increases with the logarithm of the size of the database. In fact, because the top level (or top few levels) can be cached in RAM, the disk access cost is proportional to the *size of the database as a multiple of the size of the data cached in RAM.*
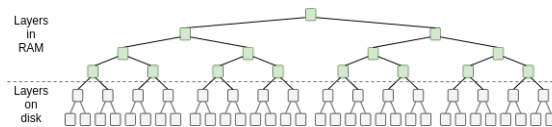
Don't take this diagram too literally; different databases work in different ways, and often the part in memory is just a single (but big) layer (see LSM trees as used in leveldb). But the basic principles are the same.

For example, if the cache is 4 GB, and we assume that each layer of the database is 4x bigger than the previous, then Ethereum's current ~64 GB state would require ~2 accesses. But if the state size increases by 4x to ~256 GB, then this would increase to ~3 accesses (so 1.5x more accesses per read). Hence, a 4x increase in the gas limit, which would increase both the state size and the number of reads, could actually translate into a ~6x increase in block verification time. The effect may be even stronger: hard disks often take longer to read and write when they are full than when they are near-empty.

So what does this mean for Ethereum?

Today in the Ethereum blockchain, running a node already is challenging for many users, though it is still at least *possible* on regular hardware (I just synced a node on my laptop while writing this post!). Hence, we are close to hitting bottlenecks. **The issue that core developers are most concerned with is storage size.** Thus, at present, valiant efforts at solving bottlenecks in computation and data, and even changes to the consensus algorithm, are unlikely to lead to large gas limit increases being accepted. Even solving Ethereum's largest outstanding DoS vulnerability only led to a gas limit increase of 20%.

The only solution to storage size problems is statelessness and state expiry. **Statelessness** allows for a class of nodes that verify the chain *without* maintaining permanent storage. **State expiry** pushes out state that has not been recently accessed, forcing users to manually provide proofs to renew it. Both of these paths have been worked at for a long time, and proof-of-concept implementation on statelessness has already started. These two improvements combined can greatly alleviate these concerns and open up room for a significant gas limit increase. **But even after statelessness and state expiry are implemented, gas limits may only increase safely by perhaps ~3x until the other limitations start to dominate.**

Another possible medium-term solution is using ZK-SNARKs to verify transactions. ZK-SNARKs would ensure that regular users do not have to personally store the state *or* verify blocks, though they still would need to download all the data in blocks to protect against data unavailability attacks. Additionally, even if attackers cannot force *invalid* blocks through, if capacity is increased to the point where running a consensus node is *too* difficult, there is still the risk of coordinated censorship attacks. Hence, ZK-SNARKs cannot increase capacity infinitely, but they still can increase capacity by a significant margin (perhaps 1-2 orders of magnitude). Some chains are exploring this approach at layer 1; Ethereum is getting the benefits of this approach through layer-2 protocols (called ZK rollups) such as zksync, Loopring and Starknet.

What happens after sharding?

Sharding fundamentally gets around the above limitations, because it decouples the data contained on a blockchain from the data that a single node needs to process and store. Instead of nodes verifying blocks by personally downloading and executing them, they use advanced mathematical and cryptographic techniques to verify blocks indirectly.

As a result, sharded blockchains can safely have very high levels of transaction throughput that non-sharded blockchains cannot. This does require a lot of cryptographic cleverness in creating efficient substitutes for naive full validation that successfully reject invalid blocks, but it can be done: the theory is well-established and proof-of-concepts based on draft specifications are already being worked on.
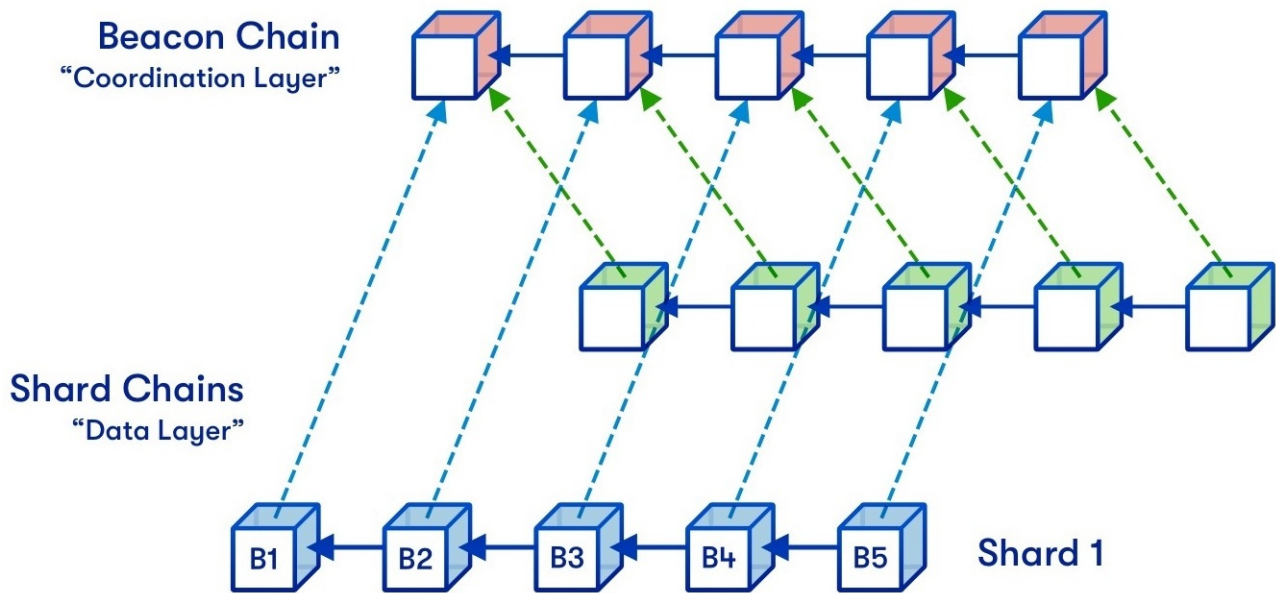
Ethereum is planning to use **quadratic sharding**, where total scalability is limited by the fact that a node has to be able to process both a single shard and the beacon chain which has to perform some fixed amount of management work for each shard. If shards are too big, nodes can no longer process individual shards, and if there are too many shards, nodes can no longer process the beacon chain. The product of these two constraints forms the upper bound.

Conceivably, one could go further by doing cubic sharding, or even exponential sharding. Data availability sampling would certainly become much more complex in such a design, but it can be done. But Ethereum is not going further than quadratic. The reason is that the extra scalability gains that you get by going from shards-of-transactions to shards-of-shards-of-transactions actually cannot be realized without other risks becoming unacceptably high.

So what are these risks?

Minimum user count

A non-sharded blockchain can conceivably run as long as there is even one user that cares to participate in it. Sharded blockchains are not like this: no single node can process the whole chain, and so you need enough nodes so that they can at least process the chain *together*. If each node can process 50 TPS, and the chain can process 10000 TPS, then the chain needs at least 200 nodes to survive. If the chain at any point gets to less than 200 nodes, then either nodes stop being able to keep up with the chain, or nodes stop being able to detect invalid blocks, or a number of other bad things may happen, depending on how the node software is set up.

In practice, the safe minimum count is several times higher than the naive "chain TPS divided by node TPS" heuristic due to the need for redundancy (including for data availability sampling); for our above example, let's call it 1000 nodes.

If a sharded blockchain's capacity increases by 10x, the minimum user count also increases by 10x. Now, you might ask: why don't we start with a little bit of capacity, and increase it only when we see lots of users so we actually need it, and decrease it if the user count goes back down?

There are a few problems with this:

1. A blockchain itself cannot reliably detect how many unique users are on it, and so this would require some kind of governance to detect and set the shard count. Governance over capacity limits can easily become a locus of division and conflict.
2. What if many users suddenly and unexpectedly drop out at the same time?
3. Increasing the minimum number of users needed for a fork to start makes it harder to defend against hostile takeovers.

A minimum user count of under 1,000 is almost certainly fine. A minimum user count of 1 million, on the other hand, is certainly not. Even a minimum user count of 10,000 is arguably starting to get risky. **Hence, it seems difficult to justify a sharded blockchain having more than a few hundred shards**.

History retrievability

An important property of a blockchain that users really value is **permanence**. A digital asset stored on a server will stop existing in 10 years when the company goes bankrupt or loses interest in maintaining that ecosystem. An NFT on Ethereum, on the other hand, *is forever*.

Yes, people will still be downloading and examining your cryptokitties in the year 2371. Deal with it.

But once a blockchain's capacity gets too high, it becomes harder to store all that data, until at some point there's a large risk that some part of history will just end up being stored by... nobody.

Quantifying this risk is easy. Take the blockchain's data capacity in MB/sec, and multiply by ~30 to get the amount of data stored in terabytes per year. The current sharding plan has a data capacity of ~1.3 MB/sec, so about 40 TB/year. If that is increased by 10x, this becomes 400 TB/year. If we want the data to be not just accessible, but accessible *conveniently*, we would also need metadata (eg. decompressing rollup transactions), so make that 4 petabytes per year, or 40 petabytes after a decade. The Internet Archive uses 50 petabytes. So that's a reasonable upper bound for how large a sharded blockchain can safely get.

Hence, it looks like on both of these dimensions, **the Ethereum sharding design is actually already roughly targeted fairly close to reasonable maximum**

safe values. The constants can be increased *a little bit*, but not too much.

Summary

There are two ways to try to scale a blockchain: **fundamental technical improvements**, and simply **increasing the parameters**. Increasing the parameters sounds very attractive at first: if you do the math on a napkin, it is easy to convince yourself that a consumer laptop can process thousands of transactions per second, no ZK-SNARKs or rollups or sharding required. Unfortunately, there are many subtle reasons why this approach is fundamentally flawed.

Computers running blockchain nodes cannot spend 100% of CPU power validating the chain; they need a large safety margin to resist unexpected DoS attacks, they need spare capacity for tasks like processing transactions in the mempool, and you don't want running a node on a computer to make that computer unusable for *any* other applications at the same time. Bandwidth similarly has overhead: a 10 MB/s connection does NOT mean you can have a 10 megabyte block every second! A 1-5 megabyte block every 12 seconds, *maybe*. And it is the same with storage. Increasing hardware requirements for running a node and limiting node-running to specialized actors is *not* a solution. **For a blockchain to be decentralized, it's crucially important for regular users to be able to run a node, and to have a culture where running nodes is a common activity**.

Fundamental technical improvements, on the other hand, *can* work. Currently, **the main bottleneck in Ethereum is storage size, and statelessness and state expiry can fix this** and allow an increase of perhaps up to ~3x - but not more, as we want running a node to become *easier* than it is today. **Sharded blockchains can scale much further**, because no single node in a sharded blockchain needs to process *every* transaction. But even there, there are limits to capacity: as capacity goes up, the minimum safe user count goes up, and the cost of archiving the chain (and the risk that data is lost if no one bothers to archive the chain) goes up. But we don't have to worry too much: those limits are high enough that we can probably process over a million transactions per second with the full security of a blockchain. But it's going to take work to do this without sacrificing the decentralization that makes blockchains so valuable.

Why sharding is great: demystifying the technical properties

2021 Apr 07

*Special thanks to Dankrad Feist and Aditya Asgaonkar for review*

Sharding is the future of Ethereum scalability, and it will be key to helping the ecosystem support many thousands of transactions per second and allowing large portions of the world to regularly use the platform at an affordable cost. However, it is also one of the more misunderstood concepts in the Ethereum ecosystem and in blockchain ecosystems more broadly. It refers to a very specific set of ideas with very specific properties, but it often gets conflated with techniques that have very different and often much weaker security properties. The purpose of this post will be to explain exactly what specific properties sharding provides, how it differs from other technologies that are *not* sharding, and what sacrifices a sharded system has to make to achieve these properties.
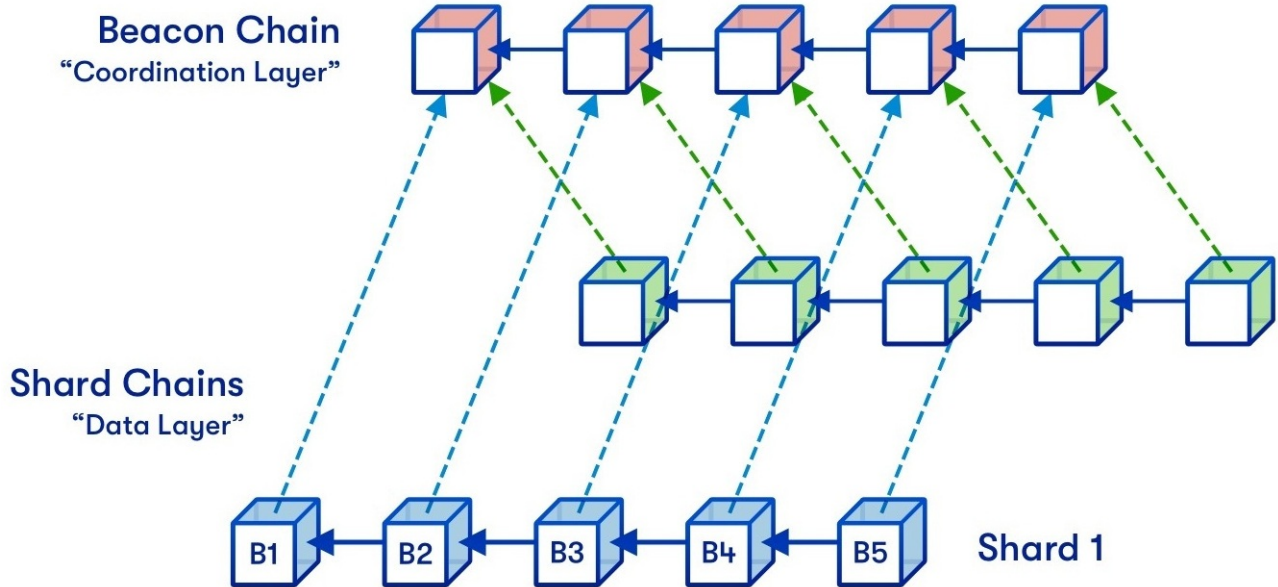
One of the many depictions of a sharded version of Ethereum. Original diagram by Hsiao-wei Wang, design by Quantstamp.

The Scalability Trilemma

The best way to describe sharding starts from the problem statement that shaped and inspired the solution: **the Scalability Trilemma**.
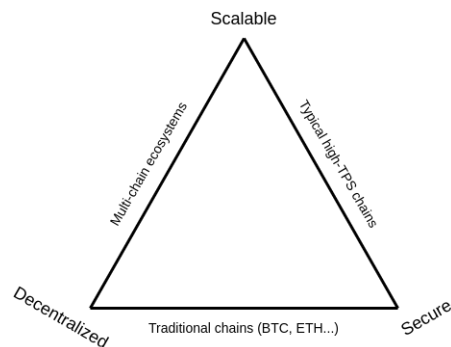
The scalability trilemma says that there are three properties that a blockchain try to have, and that, **if you stick to "simple" techniques, you can only get two of those three**. The three properties are:

- **Scalability**: the chain can process more transactions than a single regular node (think: a consumer laptop) can verify.
- **Decentralization**: the chain can run without any trust dependencies on a small group of large centralized actors. This is typically interpreted to mean that there should not be any trust (or even honest-majority assumption) of a set of nodes that you cannot join with just a consumer laptop.
- **Security**: the chain can resist a large percentage of participating nodes trying to attack it (ideally 50%; anything above 25% is fine, 5% is definitely *not* fine).

Now we can look at the three classes of "easy solutions" that only get two of the three:

- **Traditional blockchains** - including Bitcoin, pre-PoS/sharding Ethereum, Litecoin, and other similar chains. These rely on every participant running a full node that verifies every transaction, and so they have decentralization and security, but not scalability.
- **High-TPS chains** - including the DPoS family but also many others. These rely on a small number of nodes (often 10-100) maintaining consensus among themselves, with users having to trust a majority of these nodes. This is scalable and secure (using the definitions above), but it is not decentralized.
- **Multi-chain ecosystems** - this refers to the general concept of "scaling out" by having different applications live on different chains and using cross-chain-communication protocols to talk between them. This is decentralized and scalable, but it is not secure, because an attacker need only get a consensus node majority in one of the many chains (so often <1% of the whole ecosystem) to break that chain and possibly cause ripple effects that cause great damage to applications in other chains.

Sharding is a technique that gets you all three. A sharded blockchain is:

- **Scalable**: it can process far more transactions than a single node
- **Decentralized**: it can survive entirely on consumer laptops, with no dependency on "supernodes" whatsoever
- **Secure**: an attacker can't target a small part of the system with a small amount of resources; they can only try to dominate and attack the whole thing

The rest of the post will be describing how sharded blockchains manage to do this.

Sharding through Random Sampling

The easiest version of sharding to understand is sharding through random sampling. Sharding through random sampling has weaker trust properties than the forms of sharding that we are building towards in the Ethereum ecosystem, but it uses simpler technology.

The core idea is as follows. Suppose that you have a proof of stake chain with a large number (eg. 10000) validators, and you have a large number (eg. 100) blocks that need to be verified. No single computer is powerful enough to validate *all* of these blocks before the next set of blocks comes in.

Hence, what we do is we **randomly split up the work of doing the verification**. We randomly shuffle the validator list, and we assign the first 100 validators in the shuffled list to verify the first block, the second 100 validators in the shuffled list to verify the second block, etc. A randomly selected group of validators that gets assigned to verify a block (or perform some other task) is called a **committee**.
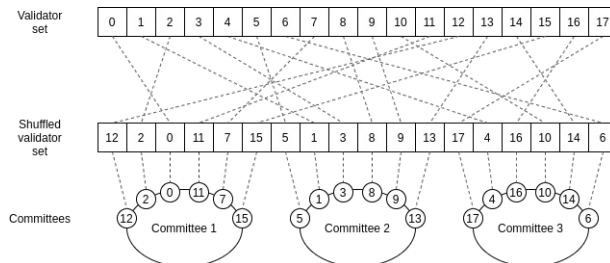
When a validator verifies a block, they publish a signature attesting to the fact that they did so. Everyone else, instead of verifying 100 entire blocks, now only verifies 10000 signatures - a much smaller amount of work, especially with BLS signature aggregation. Instead of every block being broadcasted through the same P2P network, each block is broadcasted on a different sub-network, and nodes need only join the subnets corresponding to the blocks that they are responsible for (or are interested in for other reasons).

Consider what happens if each node's computing power increases by 2x. Because each node can now safely validate 2x more signatures, you could cut the minimum staking deposit size to support 2x more validators, and so hence you can make 200 committees instead of 100. Hence, you can verify 200 blocks per slot instead of 100. Furthermore, *each individual block* could be 2x bigger. Hence, you have 2x more blocks of 2x the size, or 4x more chain capacity altogether.

We can introduce some math lingo to talk about what's going on. Using Big O notation, we use "**O(C)**" to refer to the computational capacity of a single node. A traditional blockchain can process blocks of size **O(C)**. A sharded chain as described above can process **O(C)** blocks in parallel (remember, the cost to each node to verify each block indirectly is **O(1)** because each node only needs to verify a fixed number of signatures), and each block has **O(C)** capacity, and so the sharded chain's total capacity is **O(C²)**. This is why we call this type of sharding **quadratic sharding**, and this effect is a key reason why we think that in the long run, sharding is the best way to scale a blockchain.

Frequently asked question: how is splitting into 100 committees different from splitting into 100 separate chains?

There are two key differences:

1. **The random sampling prevents the attacker from concentrating their power on one shard**. In a 100-chain multichain ecosystem, the attacker only needs ~0.5% of the total stake to wreak havoc: they can focus on 51% attacking a single chain. In a sharded blockchain, the attacker must have close to ~30-40% of the *entire* stake to do the same (in other words, the chain has **shared security**). Certainly, they can wait until they get lucky and get 51% in a single shard by random chance despite having less than 50% of the total stake, but this gets exponentially harder for attackers that have much less than 51%. If an attacker has less than ~30%, it's virtually impossible.
2. **Tight coupling: if even one shard gets a bad block, the entire chain reorgs to avoid it**. There is a social contract (and in later sections of this document we describe some ways to enforce this technologically) that a chain with even one bad block in even one shard is not acceptable and should get thrown out as soon as it is discovered. This ensures that from the point of view of an application *within* the chain, there is perfect security: contract A can rely on contract B, because if contract B misbehaves due to an attack on the chain, that entire history reverts, including the transactions in contract A that misbehaved as a result of the malfunction in contract B.

Both of these differences ensure that sharding creates an environment for applications that preserves the key safety properties of a single-chain environment, in a way that multichain ecosystems fundamentally do not.

Improving sharding with better security models

One common refrain in Bitcoin circles, and one that I completely agree with, is that **blockchains like Bitcoin (or Ethereum) do NOT completely rely on an honest majority assumption**. If there is a 51% attack on such a blockchain, then the attacker can do *some* nasty things, like reverting or censoring transactions, but they cannot insert invalid transactions. And even if they do revert or censor transactions, users running regular nodes could easily detect that behavior, so if the community wishes to coordinate to resolve the attack with a fork that takes away the attacker's power they could do so quickly.

The lack of this extra security is a key weakness of the more centralized high-TPS chains. Such chains do not, and cannot, have a culture of regular users running nodes, and so the major nodes and ecosystem players can much more easily get together and impose a protocol change that the community heavily dislikes. Even worse, the users' nodes would by default accept it. After some time, users would notice, but by then the forced protocol change would be a fait accompli: the coordination burden would be on users to reject the change, and they would have to make the painful decision to revert a day's worth or more of activity that everyone had thought was already finalized.

Ideally, we want to have a form of sharding that avoids 51% trust assumptions for validity, and preserves the powerful bulwark of security that traditional blockchains get from full verification. And this is exactly what much of our research over the last few years has been about.

Scalable verification of computation

We can break up the 51%-attack-proof scalable validation problem into two cases:

- **Validating computation**: checking that some computation was done correctly, assuming you have possession of all the inputs to the computation
- **Validating data availability**: checking that the inputs to the computation themselves are stored in some form where you can download them if you really need to; this checking should be performed *without* actually downloading the entire inputs themselves (because the data could be too large to download for every block)

Validating a block in a blockchain involves both computation and data availability checking: you need to be convinced that the transactions in the block are valid and that the new state root hash claimed in the block is the correct result of executing those transactions, but you also need to be convinced that enough data from the block was actually published so that users who download that data can compute the state and continue processing the blockchain. This second part is a very subtle but important concept called the **data availability problem**; more on this later.

Scalably validating computation is relatively easy; there are two families of techniques: **fraud proofs** and **ZK-SNARKs**.
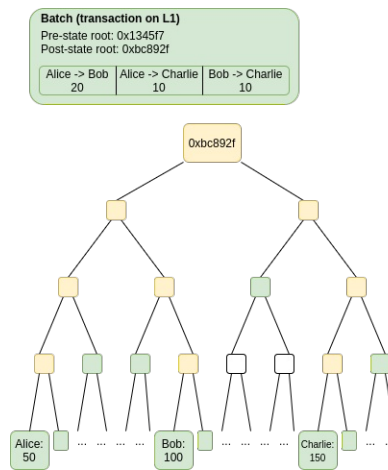
Fraud proofs are one way to verify computation scalably.

The two technologies can be described simply as follows:

- **Fraud proofs** are a system where to accept the result of a computation, you require someone with a staked **deposit** to sign a message of the form "I certify that if you make computation C with input X, you get output Y". You trust these messages by default, but you leave open the opportunity for someone else with a staked deposit to make a **challenge** (a signed message saying "I disagree, the output is Z"). Only when there is a challenge, all nodes run the computation. Whichever of the two parties was wrong loses their deposit, and all computations that depend on the result of that computation are recomputed.
- **ZK-SNARKs** are a form of *cryptographic* proof that directly proves the claim "performing computation C on input X gives output Y". The proof is cryptographically "sound": if C(x) does *not* equal Y, it's computationally infeasible to make a valid proof. The proof is also quick to verify, even if running C itself takes a huge amount of time. See this post for more mathematical details on ZK-SNARKs.

Computation based on fraud proofs is scalable because "in the normal case" you replace running a complex computation with verifying a single signature. There is the exceptional case, where you do have to verify the computation on-chain because there is a challenge, but the exceptional case is very rare because triggering it is very expensive (either the original claimer or the challenger loses a large deposit). ZK-SNARKs are conceptually simpler - they just replace a computation with a much cheaper proof verification - but the math behind how they work is considerably more complex.

There is a class of semi-scalable system which *only* scalably verifies computation, while still requiring every node to verify all the data. This can be made quite effective by using a set of compression tricks to replace most data with computation. This is the realm of rollups.

Scalable verification of data availability is harder

A fraud proof cannot be used to verify availability of data. Fraud proofs for *computation* rely on the fact that the inputs to the computation are published on-chain the moment the original claim is submitted, and so if someone challenges, the challenge execution is happening in the exact same "environment" that the original execution was happening. In the case of checking data availability, you cannot do this, because the problem is precisely the fact that there is too much data to check to publish it on chain. Hence, a fraud proof scheme for data availability runs into a key problem: someone could claim "data X is available" without publishing it, wait to get challenged, and only *then* publish data X and make the challenger appear to the rest of the network to be incorrect.

This is expanded on in **the fisherman's dilemma**:
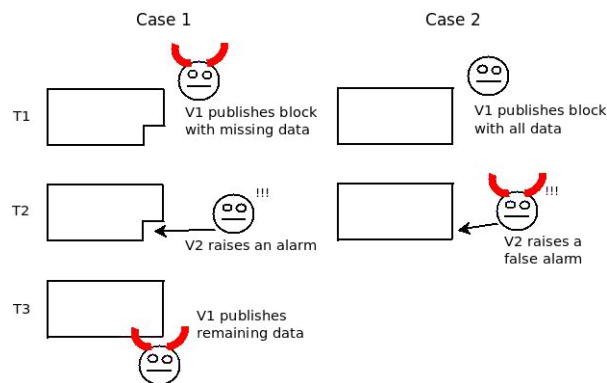
The core idea is that the two "worlds", one where V1 is an evil publisher and V2 is an honest challenger and the other where V1 is an honest publisher and V2 is an evil challenger, are indistinguishable to anyone who was not trying to download that particular piece of data at the time. And of course, in a scalable decentralized blockchain, each individual node can only hope to download a small portion of the data, so only a small portion of nodes would see anything about what went on except for the mere fact that there was a disagreement.

The fact that it is impossible to distinguish who was right and who was wrong makes it impossible to have a working fraud proof scheme for data availability.

Frequently asked question: so what if some data is unavailable? With a ZK-SNARK you can be sure everything is *valid*, and isn't that enough?

Unfortunately, mere validity is not sufficient to ensure a correctly running blockchain. This is because if the blockchain is *valid* but all the data is not *available*, then users have no way of updating the data that they need to generate proofs that any *future* block is valid. An attacker that generates a valid-but-unavailable block but then disappears can effectively stall the chain. Someone could also withhold a specific user's account data until the user pays a ransom, so the problem is not purely a liveness issue.

There are some strong information-theoretic arguments that this problem is fundamental, and there is no clever trick (eg. involving cryptographic accumulators) that can get around it. See this paper for details.

So, how do you check that 1 MB of data is available without actually trying to download it? That sounds impossible!

The key is a technology called **data availability sampling**. Data availability sampling works as follows:

1. Use a tool called **erasure coding** to expand a piece of data with N chunks into a piece of data with 2N chunks such that *any N* of those chunks can recover the entire data.
2. To check for availability, instead of trying to download the *entire* data, users simply **randomly select a constant number of positions in the block** (eg. 30

positions), and accept the block only when they have successfully found the chunks in the block at *all* of their selected positions.
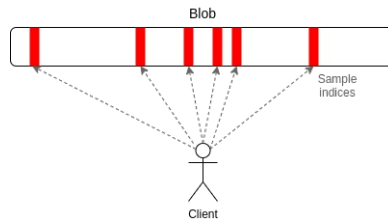
Erasure codes transform a "check for 100% availability" (every single piece of data is available) problem into a "check for 50% availability" (at least half of the pieces are available) problem. Random sampling solves the 50% availability problem. If less than 50% of the data is available, then at least one of the checks will almost certainly fail, and if at least 50% of the data is available then, while some nodes may fail to recognize a block as available, it takes only one honest node to run the erasure code reconstruction procedure to bring back the remaining 50% of the block. And so, instead of needing to download 1 MB to check the availability of a 1 MB block, you need only download a few kilobytes. This makes it feasible to run data availability checking on *every* block. See [this post](#) for how this checking can be efficiently implemented with peer-to-peer subnets.

A ZK-SNARK can be used to verify that the erasure coding on a piece of data was done *correctly*, and then Merkle branches can be used to verify individual chunks. Alternatively, you can use **polynomial commitments** (eg. [Kate (aka KZG) commitments](#)), which essentially do erasure coding *and* proving individual elements *and* correctness verification all in one simple component - and that's what Ethereum sharding is using.

Recap: how are we ensuring everything is correct again?

Suppose that you have 100 blocks and you want to efficiently verify correctness for all of them without relying on committees. We need to do the following:

- Each client performs **data availability sampling** on each block, verifying that the data in each block is available, while downloading only a few kilobytes per block even if the block as a whole is a megabyte or larger in size. A client only accepts a block when all data of their availability challenges have been correctly responded to.
- Now that we have verified data availability, it becomes easier to verify correctness. There are two techniques:
    - We can use **fraud proofs**: a few participants with staked deposits can sign off on each block's correctness. Other nodes, called **challengers** (or **fishermen**) randomly check and attempt to fully process blocks. Because we already checked data availability, it will always be possible to download the data and fully process any particular block. If they find an invalid block, they post a **challenge** that everyone verifies. If the block turns out to be bad, then that block and all future blocks that depend on that need to be re-computed.
    - We can use **ZK-SNARKs**. Each block would come with a ZK-SNARK proving correctness.
- In either of the above cases, each client only needs to do a small amount of verification work per block, no matter how big the block is. In the case of fraud proofs, occasionally blocks will need to be fully verified on-chain, but this should be extremely rare because triggering even one challenge is very expensive.

And that's all there is to it! In the case of Ethereum sharding, the near-term plan is to make sharded blocks **data-only**; that is, the shards are *purely* a "data availability engine", and it's the job of **[layer-2 rollups](#)** to use that secure data space, plus either fraud proofs or ZK-SNARKs, to implement high-throughput secure transaction processing capabilities. However, it's completely possible to create such a built-in system to add "native" high-throughput execution.

What are the key properties of sharded systems and what are the tradeoffs?

The key goal of sharding is to come as close as possible to replicating the most important security properties of traditional (non-sharded) blockchains but without the need for each node to personally verify each transaction.

Sharding comes quite close. In a traditional blockchain:

- **Invalid blocks cannot get through** because validating nodes notice that they are invalid and ignore them.
- **Unavailable blocks cannot get through** because validating nodes fail to download them and ignore them.

In a sharded blockchain with advanced security features:

- **Invalid blocks cannot get through** because either:
    - A fraud proof quickly catches them and informs the entire network of the block's incorrectness, and heavily penalizes the creator, or
    - A ZK-SNARK proves correctness, and you cannot make a valid ZK-SNARK for an invalid block.
- **Unavailable blocks cannot get through** because:
    - If less than 50% of a block's data is available, at least one data availability sample check will almost certainly fail for each client, causing the client to reject the block,
    - If at least 50% of a block's data is available, then actually the entire block is available, because it takes only a single honest node to reconstruct the rest of the block.

Traditional high-TPS chains without sharding do not have a way of providing these guarantees. Multichain ecosystems do not have a way of avoiding the problem of an attacker selecting one chain for attack and easily taking it over (the chains *could* share security, but if this was done poorly it would turn into a de-facto traditional high-TPS chain with all its disadvantages, and if it was done well, it would just be a more complicated implementation of the above sharding techniques).

Sidechains are highly implementation-dependent, but they are typically vulnerable to either the weaknesses of traditional high-TPS chains (this is if they share miners/validators), or the weaknesses of multichain ecosystems (this is if they do not share miners/validators). Sharded chains avoid these issues.

However, there are some chinks in the sharded system's armor. Notably:

- **Sharded chains that rely *only* on committees are vulnerable to adaptive adversaries, and have weaker accountability**. That is, if the adversary has the ability to hack into (or just shut down) any set of nodes of their choosing in real time, then they only need to attack a small number of nodes to break a single committee. Furthermore, if an adversary (whether an adaptive adversary or just an attacker with 50% of the total stake) does break a single committee, only a few of their nodes (the ones in that committee) can be publicly confirmed to be participating in that attack, and so only a small amount of stake can be penalized. This is another key reason why data availability sampling together with either fraud proofs or ZK-SNARKs are an important complement to random sampling techniques.
- **Data availability sampling is only secure if there is a sufficient number of online clients** that they collectively make enough data availability sampling requests that the responses almost always overlap to comprise at least 50% of the block. In practice, this means that **there must be a few hundred clients online** (and this number increases the higher the ratio of the capacity of the system to the capacity of a single node). This is a [few-of-N trust model](#) - generally quite trustworthy, but certainly not as robust as the 0-of-N trust that nodes in non-sharded chains have for availability.
- **If the sharded chain relies on fraud proofs, then it relies on timing assumptions**; if the network is too slow, nodes could accept a block as finalized before the fraud proof comes in showing that it is wrong. Fortunately, if you follow a strict rule of reverting all invalid blocks once the invalidity is discovered, this threshold is a user-set parameter: each individual user chooses how long they wait until finality and if they didn't want long enough then suffer, but more careful users are safe. Even still, this is a weakening of the user experience. **Using ZK-SNARKs to verify validity solves this.**
- **There is a much larger amount of raw data that needs to be passed around**, increasing the risk of failures under extreme networking conditions. Small amounts of data are easier to send (and easier to [safely hide](#), if a powerful government attempts to censor the chain) than larger amounts of data. Block explorers need to store more data if they want to hold the entire chain.
- Sharded blockchains depend on sharded peer-to-peer networks, and **each individual p2p "subnet" is easier to attack because it has fewer nodes**. The [subnet model used for data availability sampling](#) mitigates this because there is some redundancy between subnets, but even still there is a risk.

These are valid concerns, though in our view they are far outweighed by the *reduction in user-level centralization* enabled by allowing more applications to run on-chain instead of through centralized layer-2 services. That said, these concerns, especially the last two, are in practice the real constraint on increasing a sharded chain's throughput beyond a certain point. There is a limit to the quadraticness of quadratic sharding.

Incidentally, the growing safety risks of sharded blockchains if their throughput becomes too high are also the key reason why the effort to extend to *super-quadratic sharding* has been largely abandoned; it looks like keeping quadratic sharding *just* quadratic really is the happy medium.

Why not centralized production and sharded verification?

One alternative to sharding that gets often proposed is to have a chain that is structured like a centralized high-TPS chain, except it uses data availability sampling and sharding on top to allow verification of validity and availability.

This improves on centralized high-TPS chains as they exist today, but it's still considerably weaker than a sharded system. This is for a few reasons:

1. **It's much harder to detect censorship by block producers in a high-TPS chain**. Censorship detection requires either (i) being able to see *every* transaction and verify that there are no transactions that clearly deserve to get in that inexplicably fail to get in, or (ii) having a 1-of-N trust model in *block producers* and verifying that no blocks fail to get in. In a centralized high-TPS chain, (i) is impossible, and (ii) is harder because the small node count makes even a 1-of-N trust model more likely to break, and if the chain has a block time that is too fast for DAS (as most centralized high-TPS chains do), it's very hard to prove that a node's blocks are not being rejected simply because they are all being published too slowly.
2. If a majority of block producers and ecosystem members tries to force through an unpopular protocol change, users' clients will certainly *detect* it, but **it's much harder for the community to rebel and fork away** because they would need to spin up a new set of very expensive high-throughput nodes to maintain a chain that keeps the old rules.
3. **Centralized infrastructure is more vulnerable to censorship imposed by external actors**. The high throughput of the block producing nodes makes them very detectable and easier to shut down. It's also politically and logistically easier to censor dedicated high-performance computation than it is to go after individual users' laptops.
4. **There's a stronger pressure for high-performance computation to move to centralized cloud services**, increasing the risk that the entire chain will be run within 1-3 companies' cloud services, and hence risk of the chain going down because of many block producers failing simultaneously. A sharded chain with a culture of running validators on one's own hardware is again much less vulnerable to this.

Properly sharded systems are better as a base layer. Given a sharded base layer, you can always create a centralized-production system (eg. because you want a high-throughput domain with synchronous composability for defi) layered on top by building it as a rollup. But if you have a base layer with a dependency on centralized block production, you cannot build a more-decentralized layer 2 on top.

Gitcoin Grants Round 9: The Next Phase of Growth

2021 Apr 02

Special thanks to the Gitcoin team for feedback and diagrams.

Special note: Any criticism in these review posts of actions taken by people or organizations, especially using terms like "collusion", "bribe" and "cabal", is only in the spirit of analysis and mechanism design, and should not be taken as (especially moral) criticism of the people and organizations themselves. You're all well-intentioned and wonderful people and I love you.

Gitcoin Grants Round 9 has just finished, and as usual the round has been a success. Along with 500,000 in matching funds, $1.38 million was donated by over 12,000 contributors to 812 different projects, making this the largest round so far. Not only old projects, but also new ones, received a large amount of funding, proving the mechanism's ability to avoid entrenchment and adapt to changing circumstances. The new East Asia-specific category in the latest two rounds has also been a success, helping to catapult multiple East Asian Ethereum projects to the forefront.

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	rekt.news	1,745	$14,873	$28,372
2	DAO Square	1,525	$8,185	$19,736
3	Bankless	1,314	$14,865	$15,603
4	Ethereum Magicians	821	$10,729	$5,399
5	EthHub	871	$7,318	$5,350
6	NFThub	817	$3,538	$4,906
7	The Daily Gwei	710	$5,901	$4,878
8	The Defiant	697	$9,223	$4,286
9	DApp Chaser	730	$3,217	$4,136
10	DeFi Dad Tutorials	620	$6,517	$3,921

CLR MATCHING ROUND 9
🌏 East Asia Grants
GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	DAOSquare	1,525	$8,185	**$19,005**
2	CREAM	792	$5,195	**$5,520**
3	DAppChaser	730	$3,217	**$4,161**
4	Gas Now	545	$4,494	**$3,032**
5	EthGrounds InariDAO	624	$2,470	**$2,774**
6	ETHPlanet	599	$2,901	**$2,644**
7	reality.eth	622	$ 11,414	**$2,581**
8	Curve Swaps	654	$2,182	**$2,459**
9	Ethereum.cn	543	$3,030	**$2,290**
10	BaoBoard	457	$2,512	**$1,767**

CLR MATCHING ROUND 9
💡 Infra Grants
GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Ethereum Swarm	3,439	$31,889	**$38,865**
2	Gitcoin Grants	2,350	$27,328	**$20,553**
3	Umbra	2,235	$12,044	**$11,101**
4	Circles UBI	1,864	$11,352	**$9,393**
5	DAppNode	1,803	$15,191	**$8,785**
6	Prysm	1,463	$22,801	**$7,321**
7	Wallet Connect	1,451	$ 24,448	**$6,869**
8	beaconcha.in	1,337	$12,272	**$5,802**
9	ethers.js	1,269	$15,914	**$5,650**
10	Nethermind	1,464	$14,096	**$5,316**

CLR MATCHING ROUND 9 — dApp tech Grants — GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	EtherDrops	2,347	$21,612	$14,756
2	APY Vision	2,552	$13,987	$13,359
3	Swivel Finance	2,483	$16,007	$11,055
4	vfat.tools	2,136	$17,553	$10,151
5	ZeroPool	2,378	$11,674	$9,542
6	clr.fund	2,309	$8,959	$9,065
7	BeyondNFT	2,010	$8995	$7,076
8	BrightID	1,747	$11,342	$6,759
9	Tonic Swirl	1,809	$12,832	$6,673
10	The Commons Stack	1,685	$9,490	$6,048

However, with growing scale, round 9 has also brought out unique and unprecedented challenges. The most important among them is collusion and fraud: in round 9, over 15% of contributions were detected as being probably fraudulent. This was, of course, inevitable and expected from the start; I have actually been surprised at how long it has taken for people to start to make serious attempts to exploit the mechanism. The Gitcoin team has responded in force, and has published a blog post detailing their strategies for detecting and responding to adversarial behavior along with a general governance overview. **However, it is my opinion that to successfully limit adversarial behavior in the long run more serious reforms, with serious sacrifices, are going to be required**.

Many new, and bigger, funders

Gitcoin continues to be successful in attracting many matching funders this round. BadgerDAO, a project that describes itself as a "DAO dedicated to building products and infrastructure to bring Bitcoin to DeFi", has donated $300,000 to the matching pool - the largest single donation so far.

Other new funders include Uniswap, Stakefish, Maskbook, FireEyes, Polygon, SushiSwap and TheGraph. As Gitcoin Grants continues to establish itself as a successful home for Ethereum public goods funding, it is also continuing to attract legitimacy as a focal point for donations from projects

wishing to support the ecosystem. This is a sign of success, and hopefully it will continue and grow further. The next goal should be to get not just one-time contributions to the matching pool, but long-term commitments to repeated contributions (or even newly launched tokens donating a percentage of their holdings to the matching pool)!

Churn continues to be healthy

One long-time concern with Gitcoin Grants is the balance between stability and entrenchment: if each project's match award changes too much from round to round, then it's hard for teams to rely on Gitcoin Grants for funding, and if the match awards change too little, it's hard for new projects to get included.

We can measure this! To start off, let's compare the top-10 projects in this round to the top-10 projects in the previous round.
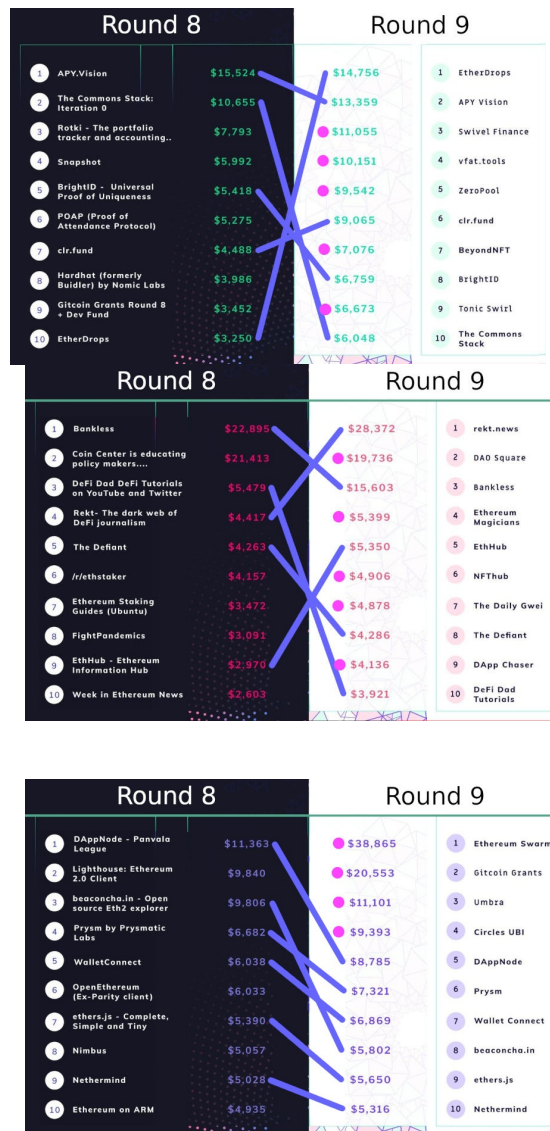
In all cases, about half of the top-10 carries over from the previous round and about half is new (the flipside, of course is that half the top-10 drops out). The charts are a slight understatement: the Gitcoin Grants dev fund and POAP appear to have dropped out but actually merely changed categories, so something like 40% churn may be a more accurate number.
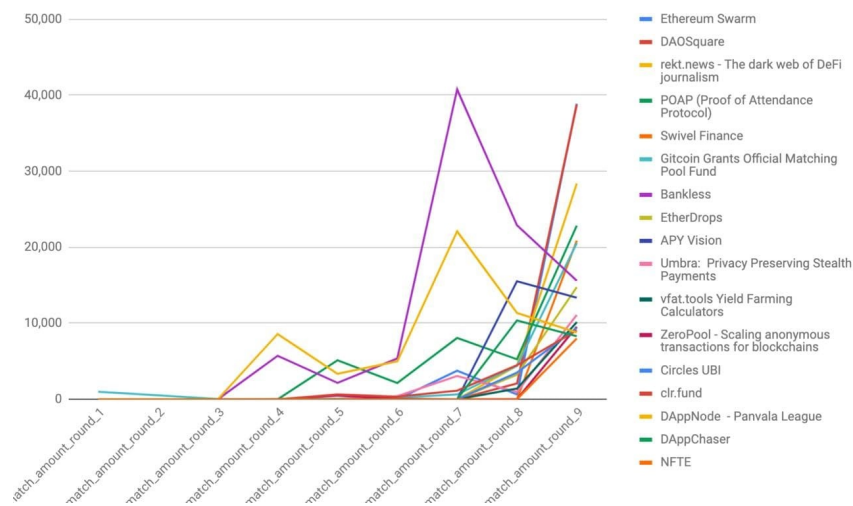
If you check the results from round 8 against [round 7](#), you also get about 50% churn, and comparing round 7 to [round 6](#) gives similar values. Hence, it is looking like the degree of churn is stable. To me, it seems like roughly 40-50% churn is a healthy level, balancing long-time projects' need for stability with the need to avoid new projects getting locked out, but this is of course only my subjective judgement.

Adversarial behavior

The challenging new phenomenon this round was the sheer scale of the adversarial behavior that was attempted. In this round, there were two major issues. First, there were large clusters of contributors discovered that were probably a few individual or small closely coordinated groups with many accounts trying to cheat the mechanism. This was discovered by proprietary analysis algorithms used by the Gitcoin team.

For this round, the Gitcoin team, in consultation with the community, decided to eat the cost of the fraud. Each project received the maximum of the match award it would receive if fraudulent transactions were accepted and the match award it would receive if they were not; the difference, about $33,000 in total, was paid out of Gitcoin's treasury. For future rounds, however, the team aims to be significantly stricter about security.
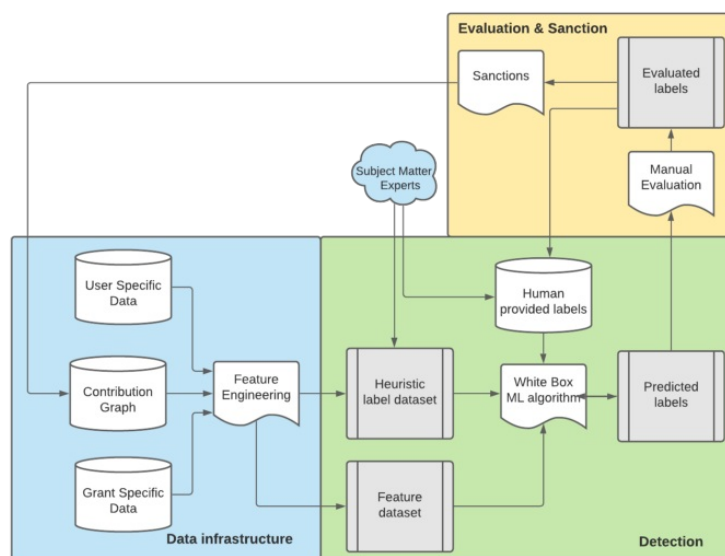
A diagram from [the Gitcoin team's post](#) describin their process for finding and dealing with adversarial behavior.

In the short term, simply ignoring fraud and accepting its costs has so far worked okay. In the long term, however, fraud must be dealt with, and this raises a challenging political concern. The algorithms that the Gitcoin team used to detect the adversarial behavior are proprietary and closed-source, and *they have to be closed-source* because otherwise the attackers could adapt and get around them. Hence, the output of the quadratic funding round is not just decided by a clear mathematical formula of the inputs. Rather, if fraudulent transactions were to be removed, it would also be fudged by what risks becoming a closed group twiddling with the outputs according to their arbitrary subjective judgements.

It is worth stressing that this is not Gitcoin's fault. Rather, what is happening is that Gitcoin has gotten big enough that it has finally bumped into the exact same problem that every social media site, no matter how well-meaning its team, has been bumping into for the past twenty years. Reddit, despite its well-meaning and open-source-oriented team, employs [many](#) secretive [tricks](#) to detect and clamp down on vote manipulation, as does every other social media site.

This is because *making algorithms that prevent undesired manipulation, but continue to do so despite the attackers themselves knowing what these algorithms are, is really hard*. In fact, **the entire science of [mechanism design](#) is a half-century-long effort to try to solve this problem**. Sometimes, there are successes. But often, they keep running into the same challenge: [collusion](#). It turns out that it's not that hard to make mechanisms that give the outcomes you want if all of the participants are acting independently, but once you admit the possibility of one individual controlling many accounts, the problem quickly becomes much harder (or even [intractable](#)).

But the fact that we can't achieve perfection doesn't mean that we can't try to come closer, and benefit from coming closer. Good mechanisms and opaque centralized intervention are substitutes: the better the mechanism, the closer to a good result the mechanism gets all by itself, and the more the secretive moderation cabal can go on vacation (an outcome that the actually-quite-friendly-and-cuddly and decentralization-loving Gitcoin moderation cabal very much wants!). In the short term, the Gitcoin team is also proactively taking a third approach: making fraud detection and response accountable by [inviting third-party analysis and community oversight](#).

Picture courtesy of the Gitcoin team's excellent blog post.

Inviting community oversight is an excellent step in preserving the mechanism's [legitimacy](#), and in paving the way for an eventual decentralization of the Gitcoin grants institution. However, it's not a 100% solution: as we've seen with technocratic organizations inside national governments, it's actually quite easy for them to retain a large amount of power despite formal democratic oversight and control. **The long-term solution is shoring up Gitcoin's *passive* security, so that active security of this type becomes less necessary.**

One important form of passive security is making some form of unique-human verification no longer optional, but instead mandatory. Gitcoin already adds the option to use phone number verification, BrightID and several other techniques to "improve an account's trust score" and get greater matching. But what Gitcoin will likely be forced to do is make it so that some verification is required *to get any matching at all*. This will be a reduction in convenience, but the effects can be mitigated by

the Gitcoin team's work on enabling more diverse and decentralized verification options, and the long-term benefit in enabling security without heavy reliance on centralized moderation, and hence getting longer-lasting legitimacy, is very much worth it.

Retroactive airdrops

A second major issue this round had to do with Maskbook. In February, Maskbook announced a token and the token distribution included a retroactive airdrop to anyone who had donated to Maskbook in previous rounds.

	Level	Regular Users	Power User
Mask Insider	NFT	Interacted but no longer hold	All Holders
	Red Packet	>=1, <=5 interaction	> 5 interactions
	ITO	>=1 interaction	
Ecosystem Participants	Voters	>=2, <=10 votes	>10 votes
	Gitcoin	Donated to >= 2 rounds (8 in total)	Mask donor before 1/1/2021
	ENS		Linked with Twitter

The table from Maskbook's announcement post showing who is eligible for the airdrops.

The controversy was that Maskbook was continuing to maintain a Gitcoin grant this round, despite now being wealthy and having set a precedent that donors to their grant might be rewarded in the future. The latter issue was particularly problematic as it *could* be construed as a form of obfuscated vote buying. **Fortunately, the situation was defused quickly; it turned out that the Maskbook team had simply forgotten to consider shutting down the grant after they released their token, and they agreed to shut it down. They are now even part of the funders' league, helping to provide matching funds for future rounds!**

Another project attempted what some construed as a "wink wink nudge nudge" strategy of obfuscated vote buying: they hinted in chat rooms that they have a Gitcoin grant and they are going to have a token. No explicit promise to reward contributors was made, but there's a case that the people reading those messages could have interpreted it as such.

In both cases, what we are seeing is that *collusion is a spectrum, not a binary*. In fact, there's a pretty wide part of the spectrum that even completely well-meaning and legitimate projects and their contributors could easily engage in.
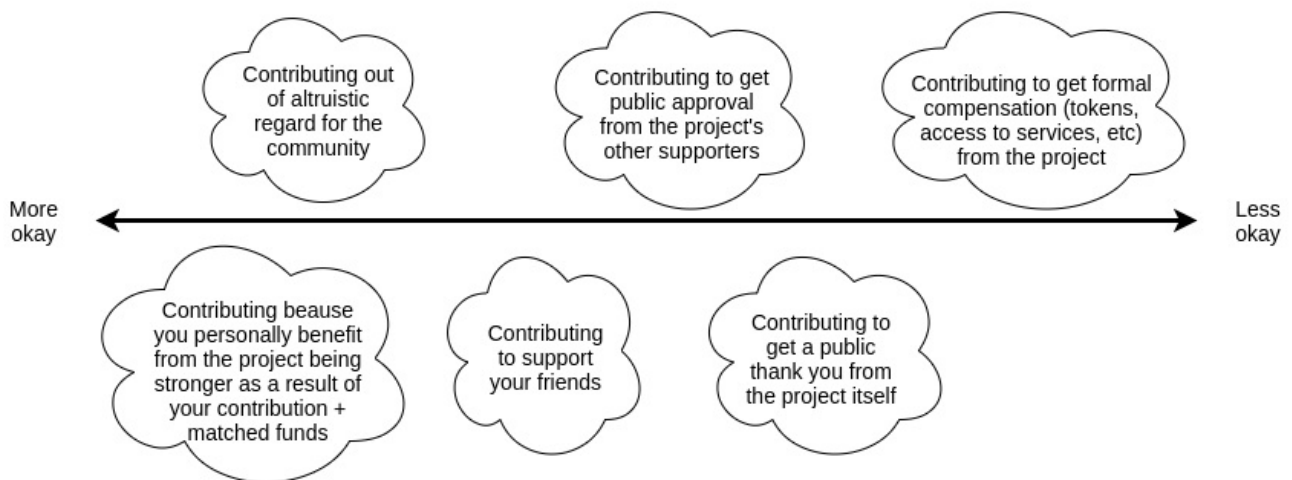
Note that this is a somewhat unusual "moral hierarchy". Normally, the more acceptable motivations would be the altruistic ones, and the less acceptable motivations would be the selfish ones. Here, though, the motivations closest to the left *and* the right are selfish; the altruistic motivation is close to the left, but it's not the *only* motivation close to the left. The key differentiator is something more subtle: **are you contributing because you like the consequences of the project getting funded (inside-the-mechanism), or are you contributing because you like some (outside-the-**

mechanism) consequences of *you personally* funding the project?

The latter motivation is problematic because it subverts the workings of quadratic funding. Quadratic funding is all about assuming that people contribute because they like the consequences of the project getting funded, recognizing that the amounts that people contribute will be much less than they ideally "should be" due to the tragedy of the commons, and mathematically compensating for that. But if there are large side-incentives for people to contribute, and these side-incentives are attached to that person specifically and so they are not reduced by the tragedy of the commons at all, then the quadratic matching *magnifies* those incentives into a very large distortion.

In both cases (Maskbook, and the other project), we saw something in the middle. The case of the other project is clear: there was an accusation that they made hints at the possibility of formal compensation, though it was not explicitly promised. In the case of Maskbook, it seems as though Maskbook did nothing wrong: the airdrop was retroactive, and so none of the contributions to Maskbook were "tainted" with impute motives. But the problem is more long-term and subtle: **if there's a *long-term pattern* of projects making retroactive airdrops to Gitcoin contributors, then users will feel a pressure to contribute primarily not to projects that they think are public goods, but rather to projects that they think are likely to later have tokens**. This subverts the dream of using Gitcoin quadratic funding to provide alternatives to token issuance as a monetization strategy.

The solution: making bribes (and retroactive airdrops) cryptographically impossible

The simplest approach would be to delist projects whose behavior comes too close to collusion from Gitcoin. In this case, though, this solution cannot work: the problem is not projects doing airdrops *while* soliciting contributions, the problem is projects doing airdrops *after* soliciting contributions. While such a project is still soliciting contributions and hence vulnerable to being delisted, there is no indication that they are planning to do an airdrop. More generally, we can see from the examples above that policing motivations is a tough challenge with many gray areas, and is generally not a good fit for the spirit of mechanism design. But if delisting and policing motivations is not the solution, then what is?

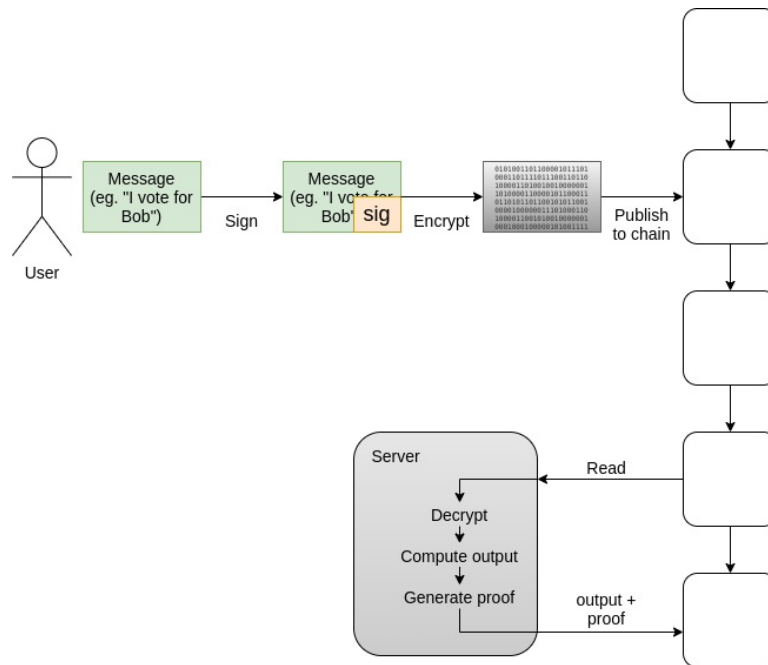The solution comes in the form of a technology called [MACI](#).

MACI is a toolkit that allows you to run *collusion-resistant applications*, which simultaneously guarantee several key properties:

- **Correctness**: invalid messages do not get processed, and the result that the mechanism outputs actually is the result of processing all valid messages and correctly computing the result.
- **Censorship resistance**: if someone participates, the mechanism cannot cheat and pretend they did not participate by selectively ignoring their messages.
- **Privacy**: no one else can see how each individual participated.
- **Collusion resistance**: a participant cannot prove to others how they participated, *even if they wanted to prove this*.

Collusion resistance is the key property: it makes bribes (or retroactive airdrops) impossible, because

users would have no way to prove that they actually contributed to someone's grant or voted for someone or performed whatever other action. This is a realization of the [secret ballot](#) concept which makes vote buying impractical today, but with cryptography.

The technical description of how this works is not that difficult. Users participate by signing a message with their private key, encrypting the signed message to a *public key* published by a central server, and publishing the encrypted signed message to the blockchain. The server downloads the messages from the blockchain, decrypts them, processes them, and outputs the result along with a [ZK-SNARK](#) to ensure that they did the computation correctly.

Users cannot prove how they participated, because they have the ability to send a "key change" message to trick anyone trying to audit them: they can first send a key change message to change their key from A to B, and then send a "fake message" signed with A. The server would reject the message, but no one else would have any way of knowing that the key change message had ever been sent. There is a trust requirement on the server, though only for privacy and coercion resistance; the server cannot publish an incorrect result either by computing incorrectly or by censoring messages. In the long term, [multi-party computation](#) can be used to decentralize the server somewhat, strengthening the privacy and coercion resistance guarantees.

There is already a quadratic funding system using MACI: [clr.fund](#). It works, though at the moment proof generation is still quite expensive; ongoing work on the project will hopefully decrease these costs soon.

Practical concerns

Note that adopting MACI does come with necessary sacrifices. In particular, there would no longer be the ability to see who contributed to what, weakening Gitcoin's "social" aspects. However, the social aspects could be redesigned and changed by taking insights from elections: elections, despite their secret ballot, frequently give out "I voted" stickers. They are not "secure" (in that a non-voter can easily get one), but they still serve the social function. One could go further while still preserving the secret ballot property: **one could make a quadratic funding setup where MACI outputs the value of *how much* each participant contributed, but not who they contributed to**. This would make it impossible for specific projects to pay people to contribute to them, but would still leave lots of space for users to express their pride in contributing. Projects could airdrop to *all* Gitcoin contributors without discriminating by project, and announce that they're doing this together with a link to their Gitcoin profile. However, users would still be able to contribute to someone else and collect the airdrop; hence, this would arguably be within bounds of fair play.

However, this is still a longer-term concern; MACI is likely not ready to be integrated for round 10. For the next few rounds, focusing on stepping up unique-human verification is still the best priority.

Some ongoing reliance on centralized moderation will be required, though hopefully this can be simultaneously reduced and made more accountable to the community. The Gitcoin team has already been taking excellent steps in this direction. And if the Gitcoin team does successfully play their role as pioneers in being the first to brave and overcome these challenges, then we will end up with a secure and scalable quadratic funding system that is ready for much broader mainstream applications!

The Most Important Scarce Resource is Legitimacy

2021 Mar 23

Special thanks to Karl Floersch, Aya Miyaguchi and Mr Silly for ideas, feedback and review.

The Bitcoin and Ethereum blockchain ecosystems both spend far more on network security - the goal of proof of work mining - than they do on everything else combined. The Bitcoin blockchain has paid an average of about *$38 million per day* in block rewards to miners since the start of the year, plus about $5m/day in transaction fees. The Ethereum blockchain comes in second, at $19.5m/day in block rewards plus $18m/day in tx fees. Meanwhile, the Ethereum Foundation's annual budget, paying for research, protocol development, grants and all sorts of other expenses, is a mere $30 million *per year*. Non-EF-sourced funding exists too, but it is at most only a few times larger. Bitcoin ecosystem expenditures on R&D are likely even lower. Bitcoin ecosystem R&D is largely funded by companies (with $250m total raised so far according to this page), and this report suggests about 57 employees; assuming fairly high salaries and many paid developers not being counted, that works out to about $20m per year.
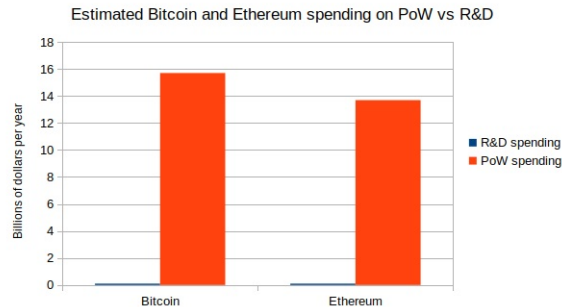
Clearly, this expenditure pattern *is a massive misallocation of resources*. The last 20% of network hashpower provides *vastly* less value to the ecosystem than those same resources would if they had gone into research and core protocol development. So why not just.... cut the PoW budget by 20% and redirect the funds to those other things instead?

The standard answer to this puzzle has to do with concepts like "public choice theory" and "Schelling fences": even though we could easily identify some valuable public goods to redirect some funding to as a one-off, making a *regular institutionalized pattern* of such decisions carries risks of political chaos and capture that are in the long run not worth it. But regardless of the reasons why, we are faced with this interesting fact that **the organisms that are the Bitcoin and Ethereum ecosystems are capable of summoning up billions of dollars of capital, but have strange and hard-to-understand restrictions on where that capital can go**.

The powerful social force that is creating this effect is worth understanding. As we are going to see, it's also the same social force behind why the Ethereum ecosystem is capable of summoning up these resources in the first place (and the technologically near-identical Ethereum Classic is not). It's also a social force that is key to helping a chain recover from a 51% attack. And it's a social force that underlies all sorts of extremely powerful mechanisms far beyond the blockchain space. For reasons that will be clear in the upcoming sections, I will give this powerful social force a name: **legitimacy**.

Coins can be owned by social contracts

To better understand the force that we are getting at, another important example is the epic saga of Steem and Hive. In early 2020, Justin Sun bought Steem-the-company, which is not the same thing as Steem-the-blockchain but did hold about 20% of the STEEM token supply. The community, naturally, did not trust Justin Sun. So they made an on-chain vote to formalize what they considered to be a longstanding "gentleman's agreement" that Steem-the-company's coins were held in trust for the common good of Steem-the-blockchain and should not be used to vote. With the help of coins held by exchanges, Justin Sun made a counterattack, and won control of enough delegates to unilaterally control the chain. The community saw no further in-protocol options. So instead they made a fork of Steem-the-blockchain, called Hive, and copied over all of the STEEM token balances - except those, including Justin Sun's, which participated in the attack.

And they got plenty of applications on board. If they had not managed this, far more users would have either stayed on Steem or moved to some different project entirely.

The lesson that we can learn from this situation is this: *Steem-the-company never actually "owned" the coins*. If they did, they would have had the practical ability to use, enjoy and abuse the coins in whatever way they wanted. But in reality, when the company tried to enjoy and abuse the coins in a way that the community did not like, *they were successfully stopped*. What's going on here is a pattern of a similar type to what we saw with the not-yet-issued Bitcoin and Ethereum coin rewards: the coins were ultimately owned not by a cryptographic key, but by some kind of *social contract*.

We can apply the same reasoning to many other structures in the blockchain space. Consider, for example, the ENS root multisig. The root multisig is controlled by seven prominent ENS and Ethereum community members. But what would happen if four of them were to come together and "upgrade" the registrar to one that transfers all the best domains to themselves? Within the context of ENS-the-smart-contract-system, they have the complete and unchallengeable ability to do this. But if they actually tried to abuse their technical ability in this way, what would happen is clear to anyone: they would be ostracized from the community, the remaining ENS community members would make a new ENS contract that restores the original domain owners, and every Ethereum application that uses ENS would repoint their UI to use the new one.

This goes well beyond smart contract structures. Why is it that Elon Musk can sell an NFT of Elon Musk's tweet, but Jeff Bezos would have a much harder time doing the same? Elon and Jeff have the same level of ability to screenshot Elon's tweet and stick it into an NFT dapp, so what's the difference? To anyone who has even a basic intuitive understanding of human social psychology (or the fake art scene), the answer is obvious: Elon selling Elon's tweet is *the real thing*, and Jeff doing the same is not. Once again, millions of dollars of value are being controlled and allocated, not by individuals or cryptographic keys, but by social conceptions of legitimacy.

And, going even further out, legitimacy governs all sorts of social status games, intellectual discourse, language, property rights, political systems and national borders. Even blockchain consensus works the same way: the only difference between a soft fork that gets accepted by the community and a 51% censorship attack after which the community coordinates an extra-protocol recovery fork to take out the attacker is legitimacy.

So what is legitimacy?

See also: my earlier post on blockchain governance.

To understand the workings of legitimacy, we need to dig down into some game theory. There are many situations in life that demand **coordinated behavior**: if you act in a certain way alone, you are likely to get nowhere (or worse), but if everyone acts together a desired result can be achieved.

	A	B
A	(5, 5)	(0, 0)
B	(0, 0)	(5, 5)

An abstract coordination game. You benefit heavily from making the same move as everyone else.

One natural example is driving on the left vs right side of the road: it doesn't really matter *what* side of the road people drive on, as long as they drive on the same side. If you switch sides at the same time as everyone else, and most people prefer the new arrangement, there can be a net benefit. But if you switch sides alone, no matter how much you prefer driving on the other side, the net result for you will be quite negative.

Now, we are ready to define legitimacy.

> **Legitimacy is a pattern of higher-order acceptance. An outcome in some social context is *legitimate* if the people in that social context broadly accept and play their part in enacting that outcome, and each individual person does so because they expect everyone else to do the same**.

Legitimacy is a phenomenon that arises naturally in coordination games. If you're not in a coordination game, there's no reason to act according to your expectation of how other people will act, and so legitimacy is not important. But as we have seen, coordination games are everywhere in society, and so legitimacy turns out to be quite important indeed. In almost any environment with coordination games that exists for long enough, there inevitably emerge some mechanisms that can choose which decision to take. These mechanisms are powered by an established culture that everyone pays attention to these mechanisms and (usually) does what they say. Each person reasons that because *everyone else* follows these mechanisms, if they do something different they will only create conflict and suffer, or at least be left in a lonely forked ecosystem all by themselves. If a mechanism successfully has the ability to make these choices, then that mechanism has legitimacy.
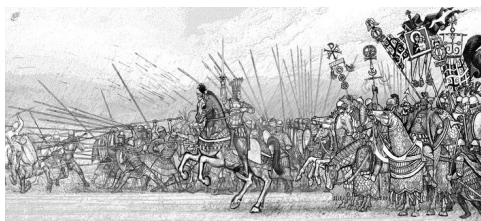
A Byzantine general rallying his troops forward. The purpose of this isn't just to make the soldiers feel brave and excited, but also to reassure them that everyone else feels brave and excited and will charge forward as well, so an individual soldier is not just committing suicide by charging forward alone.

In any context where there's a coordination game that has existed for long enough, there's likely a conception of legitimacy. **And blockchains are full of coordination games**. Which client software do you run? Which decentralized domain name registry do you ask for which address corresponds to a .eth name? Which copy of the Uniswap contract do you accept as being "the" Uniswap exchange? Even NFTs are a coordination game. The two largest parts of an NFT's value are (i) pride in holding the NFT and ability to show off your ownership, and (ii) the possibility of selling it in the future. For both of these components, it's really really important that whatever NFT you buy is recognized as *legitimate* by everyone else. In all of these cases, there's a great benefit to having the same answer as everyone else, and the mechanism that determines that equilibrium has a lot of power.

Theories of legitimacy

There are many different ways in which legitimacy can come about. In general, legitimacy arises because the thing that gains legitimacy is psychologically appealing to most people. But of course, people's psychological intuitions can be quite complex. It is impossible to make a full listing of theories of legitimacy, but we can start with a few:

- **Legitimacy by brute force**: someone convinces everyone that they are powerful enough to impose their will and resisting them will be very hard. This drives most people to submit because each person expects that *everyone else* will be too scared to resist as well.
- **Legitimacy by continuity**: if something was legitimate at time T, it is by default legitimate at time T+1.
- **Legitimacy by fairness**: something can become legitimate because it satisfies an intuitive notion of fairness. See also: my post on credible neutrality, though note that this is not the only kind of fairness.
- **Legitimacy by process**: if a process is legitimate, the outputs of that process gain legitimacy (eg. laws passed by democracies are sometimes described in this way).
- **Legitimacy by performance**: if the outputs of a process lead to results that satisfy people, then that process can gain legitimacy (eg. successful dictatorships are sometimes described in this way).
- **Legitimacy by participation**: if people participate in choosing an outcome, they are more likely to consider it legitimate. This is similar to fairness, but not quite: it rests on a psychological desire to be consistent with your previous actions.

Note that legitimacy is a descriptive concept; something can be legitimate even if you personally think that it is horrible. That said, if enough people think that an outcome is horrible, there is a higher chance that some event will happen in the future that will cause that legitimacy to go away, often at first gradually, then suddenly.

Legitimacy is a powerful social technology, and we should use it

The public goods funding situation in cryptocurrency ecosystems is fairly poor. There are hundreds of billions of dollars of capital flowing around, but public goods that are key to that capital's ongoing survival are receiving only tens of millions of dollars per year of funding.

There are two ways to respond to this fact. The first way is to be proud of these limitations and the valiant, even if not particularly effective, efforts that your community makes to work around them. This seems to be the route that the Bitcoin ecosystem often takes:

Collaborating for Philanthropy

This is why **zkSNACKs**, alongside **Francis Pouliot**, CEO of Bull Bitcoin, have come together to make a .86 bitcoin, or $40,000 contribution (split evenly between the two companies) in **support of the growth and development of Bitcoin Knots** - an open source enhanced bitcoin node/wallet software. More specifically, Bitcoin Knots is a Bitcoi full node and wallet software which can be used as an alternative to the more popular Bitcoin Core.

One of Bull Bitcoin's core values is "skin in the game".

> *Cypherpunks write code, but cypherpunks don't always get paid. We can't expect the world's most talented experts to contribute indefinitely without financial compensation. If the companies that profit from Bitcoin open-source development don't provide the necessary funding, who will? ~ Francis Pouliot*

The personal self-sacrifice of the teams funding core development is of course admirable, but it's admirable the same way that Eliud Kipchoge running a marathon in under 2 hours is admirable: it's an impressive show of human fortitude, but it's not the future of transportation (or, in this case, public goods funding). Much like we have much better technologies to allow people to move 42 km in under an hour without exceptional fortitude and years of training, **we should also focus on building better social technologies to fund public goods at the scales that we need, and as a systemic part of our economic ecology and not one-off acts of philanthropic initiative**.

Now, let us get back to cryptocurrency. A major power of cryptocurrency (and other digital assets such as domain names, virtual land and NFTs) is that it allows communities to summon up large amounts of capital without any individual person needing to personally donate that capital. However, *this capital is constrained by conceptions of legitimacy*: you cannot simply allocate it to a centralized team without compromising on what makes it valuable. While Bitcoin and Ethereum do already rely on conception of legitimacy to respond to 51% attacks, using conceptions of legitimacy to guide in-protocol funding of public goods is much harder. But at the increasingly rich application layer where new protocols are constantly being created, we have quite a bit more flexibility in where that funding could go.

Legitimacy in Bitshares

One of the long-forgotten, but in my opinion very innovative, ideas from the early cryptocurrency space was the Bitshares social consensus model. Essentially, Bitshares described itself as a community of people (PTS and AGS holders) who were willing to help collectively support an ecosystem of new projects, but for a project to be welcomed into the ecosystem, it would have to allocate 10% of its token supply to existing PTS and AGS holders.

Now, of course anyone can make a project that does not allocate any coins to PTS/AGS holders, or even fork a project that did make an allocation and take the allocation out. But, as Dan Larimer says:

> You cannot force anyone to do anything, but in this market is is all network effect. If someone comes up with a compelling implementation then you can adopt the entire PTS community for the cost of generating a new genesis block. The individual who decided to start from scratch would have to build an entire new community around his system. Considering the network effect, I suspect that the coin that honors ProtoShares will win.

This is also a conception of legitimacy: any project that makes the allocation to PTS/AGS holders will get the attention and support of the community (and it will be worthwhile for each individual community member to take an interest in the project because the rest of the community is doing so as well), and any project that does not make the allocation will not. **Now, this is certainly not a conception of legitimacy that we want to replicate verbatim - there is little appetite in the Ethereum community for enriching a small group of early adopters - but the core concept can be adapted into something much more socially valuable**.

Extending the model to Ethereum

Blockchain ecosystems, Ethereum included, value freedom and decentralization. But the public goods ecology of most of these blockchains is, regrettably, still quite authority-driven and centralized: whether it's Ethereum, Zcash or any other major blockchain, there is typically one (or at most 2-3) entities that far outspend everyone else, giving independent teams that want to build public goods few options. I call this model of public goods funding "Central Capital Coordinators for Public-goods" (CCCPs).

This state of affairs is not the fault of the organizations themselves, who are typically valiantly doing their best to support the ecosystem. Rather, it's the rules of the ecosystem that are being *unfair to that organization*, because they hold the organization to an unfairly high standard. Any single centralized organization will inevitably have blindspots and at least a few categories and teams whose value that it fails to understand; this is not because anyone involved is doing anything wrong, but because such perfection is beyond the reach of small groups of humans. So there is great value in creating a more diversified and resilient approach to public goods funding to take the pressure off any single organization.

Fortunately, we already have the seed of such an alternative! The Ethereum application-layer ecosystem exists, is growing increasingly powerful, and is already showing its public-spiritedness. Companies like Gnosis have been contributing to Ethereum client development, and various Ethereum DeFi projects have donated hundreds of thousands of dollars to the Gitcoin Grants matching pool.

Gitcoin Grants has already achieved a high level of legitimacy: its public goods funding mechanism, quadratic funding, has proven itself to be credibly neutral and effective at reflecting the community's priorities and values and plugging the holes left by existing funding mechanisms. Sometimes, top Gitcoin Grants matching recipients are even used as inspiration for grants by other and more centralized grant-giving entities. The Ethereum Foundation itself has played a key role in supporting this experimentation and diversity, incubating efforts like Gitcoin Grants, along with MolochDAO and others, that then go on to get broader community support.

We can make this nascent public goods-funding ecosystem even stronger by taking the Bitshares model, and making a modification: instead of giving the strongest community support to projects who allocate tokens to a small oligarchy who bought PTS or AGS back in 2013, **we support projects that contribute a small portion of their treasuries toward the public goods that make them and the ecosystem that they depend on possible**. And, crucially, we can deny these benefits to projects that fork an existing project and do not give back value to the broader ecosystem.

There are many ways to do support public goods: making a long-term commitment to support the Gitcoin Grants matching pool, supporting Ethereum client development (also a reasonably credibly-neutral task as there's a clear definition of what an Ethereum client *is*), or even running one's own grant program whose scope goes beyond that particular application-layer project itself. The easiest way to agree on what counts as sufficient support is to agree on how much - for example, 5% of a project's spending going to support the broader ecosystem and another 1% going to public goods that go beyond the blockchain space - and rely on good faith to choose where that funding would go.

Does the community actually have that much leverage?

Of course, there are limits to the value of this kind of community support. If a competing project (or even a fork of an existing project) gives its users a much better offering, then users are going to flock to it, regardless of how many people yell at them to instead use some alternative that they consider to be more pro-social.

But these limits are different in different contexts; sometimes the community's leverage is weak, but at other times it's quite strong. An interesting case study in this regard is the case of Tether vs DAI. Tether has many scandals, but despite this traders use Tether to hold and move around dollars all the time. The more decentralized and transparent DAI, despite its benefits, is unable to take away much of Tether's market share, at least as far as traders go. But where DAI excels is applications: Augur uses DAI, xDai uses DAI, PoolTogether uses DAI, zk.money plans to use DAI, and the list goes on. What dapps use USDT? Far fewer.

Hence, though the power of community-driven legitimacy effects is not infinite, there is nevertheless considerable room for leverage, enough to encourage projects to direct at least a few percent of their budgets to the broader ecosystem. There's even a selfish reason to participate in this equilibrium: if you were the developer of an Ethereum wallet, or an author of a podcast or newsletter, and you saw two competing projects, one of which contributes significantly to ecosystem-level public goods including yourself and one which does not, for which one would you do your utmost to help them secure more market share?

NFTs: supporting public goods beyond Ethereum

The concept of supporting public goods through value generated "out of the ether" by publicly supported conceptions of legitimacy has value going far beyond the Ethereum ecosystem. An important and immediate challenge and opportunity is NFTs. NFTs stand a great chance of significantly helping many kinds of public goods, especially of the creative variety, at least partially solve their chronic and systemic funding deficiencies.

Actually a very admirable first step.

But they could also be a missed opportunity: there is little social value in helping Elon Musk earn yet another $1 million by selling his tweet when, as far as we can tell, the money is just going to himself (and, to his credit, he eventually decided not to sell). If NFTs simply become a casino that largely benefits already-wealthy celebrities, that would be a far less interesting outcome.

Fortunately, we have the ability to help shape the outcome. Which NFTs people find attractive to buy, and which ones they do not, is a question of legitimacy: if everyone agrees that one NFT is interesting and another NFT is lame, then people will strongly prefer buying the first, because it would have both higher value for bragging rights and personal pride in holding it, and because it could be resold for more because everyone else is thinking in the same way. If the conception of legitimacy for NFTs can be pulled in a good direction, there is an opportunity to establish a solid channel of funding to artists, charities and others.

Here are two potential ideas:

1. Some institution (or even DAO) could "bless" NFTs in exchange for a guarantee that some portion of the revenues goes toward a charitable cause, ensuring that multiple groups benefit at the same time. This blessing could even come with an official categorization: is the NFT dedicated to global poverty relief, scientific research, creative arts, local journalism, open source software development, empowering marginalized communities, or something else?
2. We can work with social media platforms to make NFTs more visible on people's profiles, giving buyers a way to show the values that they committed not just their words but their hard-earned money to. This could be combined with (1) to nudge users toward NFTs that contribute to valuable social causes.

There are definitely more ideas, but this is an area that certainly deserves more active coordination and thought.

In summary

- The concept of **legitimacy (higher-order acceptance) is very powerful**. Legitimacy appears in any context where there is **coordination**, and especially on the internet, coordination is everywhere.
- There are different ways in which legitimacy comes to be: **brute force, continuity, fairness, process, performance** and **participation** are among the important ones.

- Cryptocurrency is powerful because it lets us summon up large pools of capital by collective economic will, and these pools of capital are, at the beginning, not controlled by any person. Rather, these **pools of capital are *controlled directly by concepts of legitimacy***.
- It's too risky to start doing public goods funding by printing tokens at the base layer. Fortunately, however, Ethereum has a very rich **application-layer ecosystem**, where we have much more flexibility. This is in part because there's an opportunity not just to influence existing projects, but also shape new ones that will come into existence in the future.
- **Application-layer projects that support public goods in the community should get the support of the community**, and this is a big deal. The example of DAI shows that this support really matters!
- The Etherem ecosystem cares about mechanism design and innovating at the social layer. The **Ethereum ecosystem's own public goods funding challenges are a great place to start**!
- But this goes far beyond just Ethereum itself. NFTs are one example of a large pool of capital that depends on concepts of legitimacy. **The NFT industry could be a significant boon** to artists, charities and other public goods providers far beyond our own virtual corner of the world, but **this outcome is not predetermined; it depends on active coordination and support**.

Prediction Markets: Tales from the Election

Special thanks to Jeff Coleman, Karl Floersch and Robin Hanson for critical feedback and review.

Trigger warning: I express some political opinions.

Prediction markets are a subject that has interested me for many years. The idea of allowing anyone in the public to make bets about future events, and using the odds at which these bets are made as a credibly neutral source of predicted probabilities of these events, is a fascinating application of mechanism design. Closely related ideas, like futarchy, have always interested me as innovative tools that could improve governance and decision-making. And as Augur and Omen, and more recently PolyMarket, have shown, prediction markets are a fascinating application of blockchains (in all three cases, Ethereum) as well.

And the 2020 US presidential election, it seems like prediction markets are finally entering the limelight, with blockchain-based markets in particular growing from near-zero in 2016 to millions of dollars of volume in 2020. As someone who is closely interested in seeing Ethereum applications cross the chasm into widespread adoption, this of course aroused my interest. At first, I was inclined to simply watch, and not participate myself: I am not an expert on US electoral politics, so why should I expect my opinion to be more correct than that of everyone else who was already trading? But in my Twitter-sphere, I saw more and more arguments from Very Smart People whom I respected arguing that the markets *were* in fact being irrational and I *should* participate and bet against them if I can. Eventually, I was convinced.

I decided to make an experiment on the blockchain that I helped to create: I bought $2,000 worth of NTRUMP (tokens that pay $1 if Trump loses) on Augur. Little did I know then that my position would eventually increase to $308,249, earning me a profit of over $56,803, and that I would make all of these remaining bets, against willing counterparties, *after Trump had already lost the election*. What would transpire over the next two months would prove to be a fascinating case study in social psychology, expertise, arbitrage, and the limits of market efficiency, with important ramifications to anyone who is deeply interested in the possibilities of economic institution design.

Before the Election

My first bet on this election was actually not on a blockchain at all. When Kanye announced his presidential bid in July, a political theorist whom I ordinarily quite respect for his high-quality and original thinking immediately claimed on Twitter that he was confident that this would split the anti-Trump vote and lead to a Trump victory. I remember thinking at the time that this particular opinion of his was over-confident, perhaps even a result of over-internalizing the heuristic that if a viewpoint seems clever and contrarian then it is likely to be correct. So of course I offered to make a $200 bet, myself betting the boring conventional pro-Biden view, and he honorably accepted.

The election came up again on my radar in September, and this time it was the prediction markets that caught my attention. The markets gave Trump a nearly 50% chance of winning, but I saw many Very Smart People in my Twitter-sphere whom I respected pointing out that this number seemed far too high. This of course led to the familiar "efficient markets debate": if you can buy a token that gives you $1 if Trump loses for $0.52, and Trump's actual chance of losing is much higher, why wouldn't people just come in and buy the token until the price rises more? And if nobody has done this, who are you to think that you're smarter than everyone else?

Ne0liberal's Twitter thread just before Election Day does an excellent job summarizing his case against prediction markets being accurate at that time. In short, the (non-blockchain) prediction markets that most people used at least prior to 2020 have all sorts of restrictions that make it difficult for people to participate with more than a small amount of cash. As a result, if a very smart individual or a professional organization saw a probability that they believed was wrong, they would only have a very limited ability to push the price in the direction that they believe to be correct.

The most important restrictions that the paper points out are:

- Low limits (well under $1,000) on how much each person can bet
- High fees (eg. a 5% withdrawal fee on PredictIt)

And this is where I pushed back against ne0liberal in September: although the stodgy old-world centralized prediction markets may have low limits and high fees, the crypto markets do not! On Augur or Omen, there's no limit to how much someone can buy or sell if they think the price of some outcome token is too low or too high. And the blockchain-based prediction markets were following the same prices as PredictIt. If the markets really were over-estimating Trump because high fees and low trading limits were preventing the more cool-headed traders from outbidding the overly optimistic ones, then why would blockchain-based markets, which don't have those issues, show the same prices?

PredictIt   Augur

The main response my Twitter friends gave to this was that blockchain-based markets are highly niche, and very few people, particularly very few people who know much about politics, have easy access to cryptocurrency. That seemed plausible, but I was not *too* confident in that argument. And so I bet $2,000 against Trump and went no further.

The Election

Then the election happened. After an initial scare where Trump at first won more seats than we expected, Biden turned out to be the eventual winner. Whether or not the election itself validated or refuted the efficiency of prediction markets is a topic that, as far as I can tell, is quite open to interpretation. On the one hand, by a standard Bayes rule application, I should decrease my confidence of prediction markets, at least relative to Nate Silver. Prediction markets gave a 60% chance of Biden winning, Nate Silver gave a 90% chance of Biden winning. Since Biden in fact won, this is one piece of evidence that I live in a world where Nate gives the more correct answers.

But on the other hand, you can make a case that the prediction markets bettter estimated the *margin* of victory. The median of Nate's probability distribution was somewhere around 370 of 538 electoral college votes going to Biden:
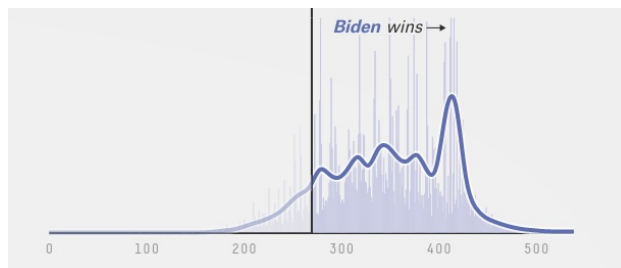
The Trump markets didn't give a probability distribution, but if you *had to* guess a probability distribution from the statistic "40% chance Trump will win", you would probably give one with a median somewhere around 300 EC votes for Biden. The actual result: 306. So the net score for prediction markets vs Nate seems to me, on reflection, ambiguous.

After the election

But what I could not have imagined at the time was that the election itself was just the beginning. A few days after the election, Biden was declared the winner by various major organizations and even a few foreign governments. Trump mounted various legal challenges to the election results, as was expected, but each of these challenges quickly failed. But for over a month, *the price of the NTRUMP tokens stayed at 85 cents*!

At the beginning, it seemed reasonable to guess that Trump had a 15% chance of overturning the results; after all, he had appointed three judges to the Supreme Court, at a time of heightened partisanship where many have come to favor team over principle. Over the next three weeks, however, it became more and more clear that the challenges were failing, and Trump's hopes continued to look grimmer with each passing day, but the NTRUMP price did not budge; in fact, it even briefly *decreased* to around $0.82. On December 11, more than five weeks after the election, the Supreme Court decisively and unanimously rejected Trump's attempts to overturn the vote, and the NTRUMP price finally rose.... to $0.88.

It was in November that I was finally convinced that the market skeptics were right, and I plunged in and bet against Trump myself. The decision was not so much about the money; after all, barely two months later I would earn and donate to GiveDirectly a far larger amount simply from holding dogecoin. Rather, it was to take part in the experiment not just as an observer, but as an active participant, and improve my personal understanding of why everyone else hadn't already plunged in to buy NTRUMP tokens before me.

Dipping in

I bought my NTRUMP on Catnip, a front-end user interface that combines together the Augur prediction market with Balancer, a Uniswap-style constant-function market maker. Catnip was by far the easiest interface for making these trades, and in my opinion contributed significantly to Augur's usability.

There are two ways to bet against Trump with Catnip:

1. Use DAI to buy NTRUMP on Catnip directly
2. Use Foundry to access an Augur feature that allows you to convert 1 DAI into 1 NTRUMP + 1 YTUMP + 1ITRUMP (the "I" stands for "invalid", more on this later), and sell the YTRUMP on Catnip

At first, I only knew about the first option. But then I discovered that Balancer has far more liquidity for YTRUMP, and so I switched to the second option.

There was also another problem: I did not have any DAI. I had ETH, and I could have sold my ETH to get DAI, but I did not want to sacrifice my ETH exposure; it would have been a shame if I earned $50,000 betting against Trump but simultaneously lost $500,000 missing out on ETH price changes. So I decided to keep my ETH price exposure the same by opening up a collateralized debt position (CDP, now also called a "vault") on MakerDAO.

A CDP is how all DAI is generated: users deposit their ETH into a smart contract, and are allowed to withdraw an amount of newly-generated DAI up to 2/3 of the value of ETH that they put in. They can get their ETH back by sending back the same amount of DAI that they withdrew plus an extra interest fee (currently 3.5%). If the value of the ETH collateral that you deposited drops to less than 150% the value of the DAI you withdrew, anyone can come in and "liquidate" the vault, forcibly selling the ETH to buy back the DAI and charging you a high penalty. Hence, it's a good idea to have a high collateralization ratio in case of sudden price movements; I had over $3 worth of ETH in my CDP for every $1 that I withdrew.

Recapping the above, here's the pipeline in diagram form:
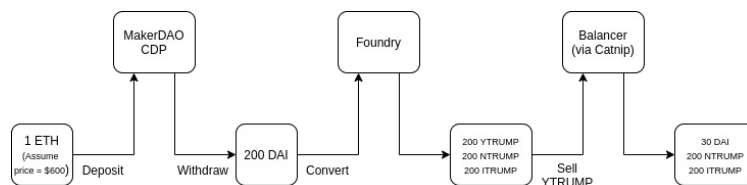
I did this many times; the slippage on Catnip meant that I could normally make trades only up to about $5,000 to $10,000 at a time without prices becoming too unfavorable (when I had skipped Foundry and bought NTRUMP with DAI directly, the limit was closer to $1,000). And after two months, I had accumulated over 367,000 NTRUMP.

Why not everyone else?

Before I went in, I had four main hypotheses about why so few others were buying up dollars for 85 cents:

1. Fear that either the Augur smart contracts would break or Trump supporters would manipulate the oracle (a [decentralized mechanism](#) where holders of Augur's REP token vote by staking their tokens on one outcome or the other) to make it return a false result
2. Capital costs: to buy these tokens, you have to lock up funds for over two months, and this removes your ability to spend those funds or make other profitable trades for that duration
3. It's too technically complicated for almost everyone to trade
4. There just really are far fewer people than I thought who are actually motivated enough to take a weird opportunity even when it presents them straight in the face

All four have reasonable arguments going for them. Smart contracts breaking [is](#) [a](#) [real](#) [risk](#), and the Augur oracle had not before been tested in such a contentious environment. Capital costs are real, and while betting against something is easier in a prediction market than in a stock market because you know that prices will never go above $1, locking up capital nevertheless competes with other lucrative opportunities in the crypto markets. Making transactions things in dapps *is* technically complicated, and it's rational to have some degree of fear-of-the-unknown.

But my experience actually going into the financial trenches, and watching the prices on this market evolve, taught me a lot about each of these hypotheses.

Fear of smart contract exploits

At first, I thought that "fear of smart contract exploits" must have been a significant part of the explanation. But over time, I have become more convinced that it is probably *not* a dominant factor. One way to see why I think this is the case is to compare the prices for YTRUMP and ITRUMP. ITRUMP stands for "Invalid Trump"; "Invalid" is an event outcome that is intended to be triggered in [some exceptional cases](#): when the description of the event is ambiguous, when the outcome of the event is not yet known when the market is resolved, when the market is unethical (eg. assassination markets), and a few other similar situations. In this market, the price of ITRUMP consistently stayed under $0.02. If someone wanted to earn a profit by attacking the market, it would be far more lucrative for them to not buy YTRUMP at $0.15, but instead buy ITRUMP at $0.02. If they buy a large amount of ITRUMP, they could earn a 50x return if they can force the "invalid" outcome to actually trigger. So if you fear an attack, buying ITRUMP is by far the most rational response. And yet, very few people did.

A further argument against fear of smart contract exploits, of course, is the fact that in every crypto application *except* prediction markets (eg. Compound, the various yield farming schemes) people are surprisingly blasé about smart contract risks. If people are willing to put their money into all sorts of risky and untested schemes even for a promise of mere 5-8% annual gains, why would they suddenly become over-cautious here?

Capital costs

Capital costs - the inconvenience and opportunity cost of locking up large amounts of money - are a challenge that I have come to appreciate much more than I did before. Just looking at the Augur side of things, I needed to lock up 308,249 DAI for an average of about two months to make a $56,803 profit. This works out to about a 175% annualized interest rate; so far, quite a good deal, even compared to the various yield farming crazes of the summer of 2020. But this becomes worse when you take into account what I needed to do on MakerDAO. Because I wanted to keep my exposure to ETH the same, I needed to get my DAI through a CDP, and safely using a CDP required a collateral ratio of over 3x. Hence, the total amount of capital I *actually* needed to lock up was somewhere around a million dollars.
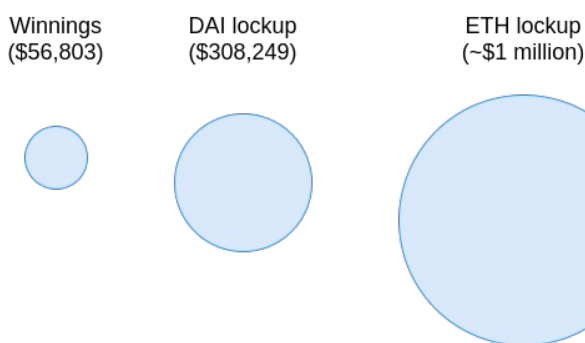
Now, the interest rates are looking less favorable. And if you add to that the possibility, however remote, that a smart contract hack, or a truly unprecedented political event, actually *will* happen, it looks less favorable still.

But even still, assuming a 3x lockup and a 3% chance of Augur breaking (I had bought ITRUMP to cover the possibility that it breaks in the "invalid" direction, so I needed only worry about the risk of breaks in the "yes" direction or the the funds being stolen outright), that works out to a risk-neutral rate of about 35%, and even lower once you take real human beings' views on risk into account. The deal is still very attractive, but on the other hand, it now looks very understandable that such numbers are unimpressive to people who live and breathe cryptocurrency with its frequent 100x ups and downs.

Trump *supporters*, on the other hand, faced none of these challenges: they cancelled out my $308,249 bet by throwing in a mere $60,000 (my winnings are less than this because of fees). When probabilities are close to 0 or 1, as is the case here, the game is *very* lopsided in favor of those who are trying to push the probability away from the extreme value. And this explains not just Trump; it's also the reason why all sorts of popular-among-a-niche candidates with no real chance of victory frequently get winning probabilities as high as 5%.

Technical complexity

I had at first tried buying NTRUMP on Augur, but technical glitches in the user interface prevented me from being able to make orders on Augur directly

(other people I talked to did not have this issue... I am still not sure what happened there). Catnip's UI is much simpler and worked excellently. However, automated market makers like Balancer (and Uniswap) work best for smaller trades; for larger trades, the slippage is too high. This is a good microcosm of the broader "AMM vs order book" debate: AMMs are more convenient but order books really do work better for large trades. Uniswap v3 is introducing an AMM design that has better capital efficiency; we shall see if that improves things.

There were other technical complexities too, though fortunately they all seem to be easily solvable. There is no reason why an interface like Catnip could not integrate the "DAI -> Foundry -> sell YTRUMP" path into a contract so that you could buy NTRUMP that way in a single transaction. In fact, the interface could even check the price and liquidity properties of the "DAI -> NTRUMP" path and the "DAI -> Foundry -> sell YTRUMP" path and give you the better trade automatically. Even withdrawing DAI from a MakerDAO CDP can be included in that path. My conclusion here is optimistic: technical complexity issues were a real barrier to participation this round, but things will be much easier in future rounds as technology improves.

Intellectual underconfidence

And now we have the final possibility: that many people (and smart people in particular) have a pathology that they suffer from excessive humility, and too easily conclude that if no one else has taken some action, then there must therefore be a good reason why that action is not worth taking.

Eliezer Yudkowsky spends the second half of his excellent book Inadequate Equilibria making this case, arguing that too many people overuse "modest epistemology", and we should be much more willing to act on the results of our reasoning, even when the result suggests that the great majority of the population is irrational or lazy or wrong about something. When I read those sections for the first time, I was unconvinced; it seemed like Eliezer was simply being overly arrogant. But having gone through this experience, I have come to see some wisdom in his position.

This was not my first time seeing the virtues of trusting one's own reasoning first hand. When I had originally started working on Ethereum, I was at first beset by fear that there must be some very good reason the project was doomed to fail. A fully programmable smart-contract-capable blockchain, I reasoned, was clearly such a great improvement over what came before, that surely many other people must have thought of it before I did. And so I fully expected that, as soon as I publish the idea, many very smart cryptographers would tell me the very good reasons why something like Ethereum was fundamentally impossible. And yet, no one ever did.

Of course, not everyone suffers from excessive modesty. Many of the people making predictions *in favor* of Trump winning the election were arguably fooled by their own excessive contrarianism. Ethereum benefited from my youthful suppression of my own modesty and fears, but there are plenty of other projects that could have benefited from more intellectual humility and avoided failures.

Not a sufferer of excessive modesty.

But nevertheless it seems to me more true than ever that, as goes the famous Yeats quote, "the best lack all conviction, while the worst are full of passionate intensity." Whatever the faults of overconfidence or contrarianism sometimes may be, it seems clear to me that spreading a society-wide message that the solution is to simply trust the existing outputs of society, whether those come in the form of academic institutions, media, governments or markets, is *not* the solution. All of these institutions can only work precisely because of the presence of individuals who think that they do not work, or who at least think that they can be wrong at least some of the time.

Lessons for futarchy

Seeing the importance of capital costs and their interplay with risks first hand is also important evidence for judging systems like **futarchy**. Futarchy, and "decision markets" more generally are an important and potentially very socially useful application of prediction markets. There is not much social value in having slightly more accurate predictions of who will be the next president. But there *is* a lot of social value in having **conditional predictions**: *if we do A, what's the chance it will lead to some good thing X, and if we do B instead what are the chances then*? Conditional predictions are important because they do not just satisfy our curiosity; they can also help us make decisions.

Though electoral prediction markets are much less useful than conditional predictions, they can help shed light on an important question: how robust are they to manipulation or even just biased and wrong opinions? We can answer this question by looking at how difficult arbitrage is: suppose that a conditional prediction market currently gives probabilities that (in your opinion) are *wrong* (could be because of ill-informed traders or an explicit manipulation attempt; we don't really care). How much of an impact can you have, and how much profit can you make, by setting things right?

Let's start with a concrete example. Suppose that we are trying to use a prediction market to choose between decision A and decision B, where each decision has some probability of achieving some desirable outcome. Suppose that *your opinion* is that decision A has a 50% chance of achieving the goal, and decision B has a 45% chance. The market, however, (in your opinion wrongly) thinks decision B has a 55% chance and decision A has a 40% chance.

Probability of good outcome if we choose strategy...	Current market position	Your opinion
A	40%	50%
B	55%	45%

Suppose that you are a small participant, so your individual bets won't affect the outcome; only many bettors acting together could. How much of your money should you bet?

The standard theory here relies on the [Kelly criterion](). Essentially, you should act to maximize the expected logarithm of your assets. In this case, we can solve the resulting equation. Suppose you invest portion r of your money into buying A-token for $0.4. Your expected new log-wealth, from your point of view, would be:

$(0.5 * \log((1-r) + \frac{r}{0.4}) + 0.5 * \log(1-r))$

The first term is the 50% chance (from your point of view) that the bet pays off, and the portion r that you invest grows by 2.5x (as you bought dollars at 40 cents). The second term is the 50% chance that the bet does not pay off, and you lose the portion you bet. We can use calculus to find the r that maximizes this; for the lazy, [here's WolframAlpha](). The answer is $r = \frac{1}{6}$. If other people buy and the price for A on the market gets up to 47% (and B gets down to 48%), we can redo the calculation for the last trader who would flip the market over to make it correctly favor A:

$(0.5 * \log((1-r) + \frac{r}{0.47}) + 0.5 * \log(1-r))$

Here, the expected-log-wealth-maximizing r is a mere 0.0566. The conclusion is clear: when decisions are close and when there is a lot of noise, it turns out that it only makes sense to invest a small portion of your money in a market. And this is assuming rationality; most people invest *less* into uncertain gambles than the Kelly criterion says they should. Capital costs stack on top even further. But if an attacker *really* wants to force outcome B through because they want it to happen for personal reasons, they can simply put *all* of their capital toward buying that token. All in all, the game can easily be lopsided more than 20:1 in favor of the attacker.

Of course, in reality attackers are rarely willing to stake all their funds on one decision. And futarchy is not the only mechanism that is vulerable to attacks: stock markets are similarly vulnerable, and non-market decision mechanisms can also be manipulated by determined wealthy attackers in all sorts of ways. But nevertheless, we should be wary of assuming that futarchy will propel us to new heights of decision-making accuracy.

Interestingly enough, the math seems to suggest that futarchy would work best when the expected manipulators would want to push the outcome *toward* an extreme value. An example of this might be liability insurance, as someone wishing to improperly obtain insurance would effectively be trying to force the market-estimated probability that an unfavorable event will happen down to zero. And as it turns out, liability insurance is futarchy inventor Robin Hanson's [new favorite policy prescription]().

Can prediction markets become better?

The final question to ask is: are prediction markets doomed to repeat errors as grave as giving Trump a 15% chance of overturning the election in early December, and a 12% chance of overturning it even after the Supreme Court including three judges whom he appointed telling him to screw off? Or could the markets improve over time? My answer is, surprisingly, emphatically on the optimistic side, and I see a few reasons for optimism.

Markets as natural selection

First, these events have given me a new perspective on how market efficiency and rationality might actually come about. Too often, proponents of market efficiency theories claim that market efficiency results because most participants are rational (or at least the rationals outweigh any coherent group of deluded people), and this is true as an axiom. But instead, we could take an *evolutionary* perspective on what is going on.

Crypto is a young ecosystem. It is an ecosystem that is still quite disconnected from the mainstream, Elon's recent tweets notwithstanding, and that does not yet have much expertise in the minutiae of electoral politics. Those who *are* experts in electoral politics have a hard time getting into crypto, and crypto has a large presence of not-always-correct forms of contrarianism especially when it comes to politics. But what happened this year is that within the crypto space, prediction market users who correctly expected Biden to win got an 18% increase to their capital, and prediction market users who incorrectly expected Trump to win got a 100% decrease to their capital (or at least the portion they put into the bet).
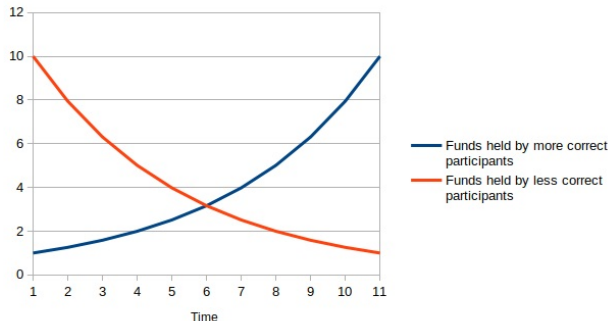
Thus, there is a *selection pressure* in favor of the type of people who make bets that turn out to be correct. After ten rounds of this, good predictors will have more capital to bet with, and bad predictors will have less capital to bet with. This does *not* rely on anyone "getting wiser" or "learning their lesson" or any other assumption about humans' capacity to reason and learn. It is simply a result of selection dynamics that over time, participants that are good at making correct guesses will come to dominate the ecosystem.

Note that prediction markets fare better than stock markets in this regard: the "nouveau riche" of stock markets often arise from getting lucky on a single thousandfold gain, adding a lot of noise to the signal, but in prediction markets, prices are bounded between 0 and 1, limiting the impact of any one single event.

Better participants and better technology

Second, prediction markets themselves will improve. User interfaces have greatly improved already, and will continue to improve further. The complexity of the MakerDAO -> Foundry -> Catnip cycle will be abstracted away into a single transaction. Blockchain scaling technology will improve, reducing fees for participants (The [ZK-rollup]() [Loopring]() with a built-in AMM is already live on the Ethereum mainnet, and a prediction market could theoretically run on it).

Third, the demonstration that we saw of the prediction market working correctly will ease participants' fears. Users will see that the Augur oracle is capable of giving correct outputs even in very contentious situations (this time, there were two rounds of disputes, but the no side nevertheless cleanly won). People from outside the crypto space will see that the process works and be more inclined to participate. Perhaps even Nate Silver himself might get some DAI and use Augur, Omen, Polymarket and other markets to supplement his income in 2022 and beyond.

Fourth, prediction market tech itself could improve. [Here is a proposal from myself]() on a market design that could make it more capital-efficient to simultaneously bet against many unlikely events, helping to prevent unlikely outcomes from getting irrationally high odds. Other ideas will surely spring up, and I look forward to seeing more experimentation in this direction.

Conclusion

This whole saga has proven to be an incredibly interesting direct trial-by-first test of prediction markets and how they collide with the complexities of

individual and social psychology. It shows a lot about how market efficiency actually works in practice, what are the limits of it and what could be done to improve it.

It has also been an excellent demonstration of the power of blockchains; in fact, it is one of the Ethereum applications that have provided to me the most concrete value. Blockchains are often criticized for being speculative toys and not doing anything meaningful except for self-referential games (tokens, with yield farming, whose returns are powered by... the launch of other tokens). There are certainly exceptions that the critics fail to recognize; I personally have benefited from ENS and even from using ETH for payments on several occasions where all credit card options failed. But over the last few months, it seems like we have seen a rapid burst in Ethereum applications being concretely useful for people and interacting with the real world, and prediction markets are a key example of this.

I expect prediction markets to become an increasingly important Ethereum application in the years to come. The 2020 election was only the beginning; I expect more interest in prediction markets going forward, not just for elections but for conditional predictions, decision-making and other applications as well. The amazing promises of what prediction markets could bring if they work mathematically optimally will, of course, continue to collide with the limits of human reality, and hopefully, over time, we will get a much clearer view of exactly where this new social technology can provide the most value.

An approximate introduction to how zk-SNARKs are possible

2021 Jan 26

Special thanks to Dankrad Feist, Karl Floersch and Hsiao-wei Wang for feedback and review.

Perhaps the most powerful cryptographic technology to come out of the last decade is general-purpose succinct zero knowledge proofs, usually called zk-SNARKs ("zero knowledge succinct arguments of knowledge"). A zk-SNARK allows you to generate a proof that some computation has some particular output, in such a way that the proof can be verified extremely quickly even if the underlying computation takes a very long time to run. The "ZK" ("zero knowledge") part adds an additional feature: the proof can keep some of the inputs to the computation hidden.

For example, you can make a proof for the statement "I know a secret number such that if you take the word 'cow', add the number to the end, and SHA256 hash it 100 million times, the output starts with `0x57d00485aa`". The verifier can verify the proof far more quickly than it would take for them to run 100 million hashes themselves, and the proof would also not reveal what the secret number is.

In the context of blockchains, this has two very powerful applications:

1. **Scalability**: if a block takes a long time to verify, one person can verify it and generate a proof, and everyone else can just quickly verify the proof instead
2. **Privacy**: you can prove that you have the right to transfer some asset (you received it, and you didn't already transfer it) without revealing the link to which asset you received. This ensures security without unduly leaking information about who is transacting with whom to the public.

But zk-SNARKs are quite complex; indeed, as recently as in 2014-17 they were still frequently called "moon math". The good news is that since then, the protocols have become simpler and our understanding of them has become much better. This post will try to explain how ZK-SNARKs work, in a way that should be understandable to someone with a medium level of understanding of mathematics.

Note that we will focus on scalability; privacy for these protocols is actually relatively easy once the scalability is there, so we will get back to that topic at the end.

Why ZK-SNARKs "should" be hard

Let us take the example that we started with: we have a number (we can encode "cow" followed by the secret input as an integer), we take the SHA256 hash of that number, then we do that again another 99,999,999 times, we get the output, and we check what its starting digits are. This is a *huge* computation.

A "succinct" proof is one where both the size of the proof and the time required to verify it grow much more slowly than the computation to be verified. If we want a "succinct" proof, we cannot require the verifier to do some work *per round of hashing* (because then the verification time would be proportional to the computation). Instead, the verifier must somehow check the whole computation without peeking into each individual piece of the computation.

One natural technique is *random sampling*: how about we just have the verifier peek into the computation in 500 different places, check that those parts are correct, and if all 500 checks pass then assume that the rest of the computation must with high probability be fine, too?

Such a procedure could even be turned into a non-interactive proof using the **Fiat-Shamir heuristic**: the prover computes a Merkle root of the computation, uses the Merkle root to pseudorandomly choose 500 indices, and provides the 500 corresponding Merkle branches of the data. The key idea is that the prover does not know which branches they will need to reveal until they have already "committed to" the data. If a malicious prover tries to fudge the data after learning which indices are going to be checked, that would change the Merkle root, which would result in a new set of random indices, which would require fudging the data again... trapping the malicious prover in an endless cycle.

But unfortunately there is a fatal flaw in naively applying random sampling to spot-check a computation in this way: computation is inherently *fragile*. If a malicious prover flips one bit somewhere in the middle of a computation, they can make it give a completely different result, and a random sampling verifier would almost never find out.
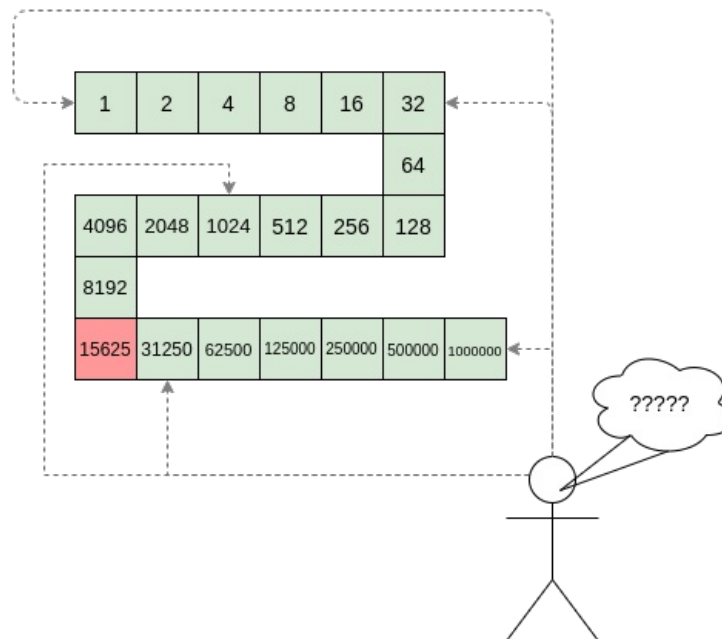
It only takes one deliberately inserted error, that a random check would almost never catch, to make a computation give a completely incorrect result.

If tasked with the problem of coming up with a zk-SNARK protocol, many people would make their way to this point and then get stuck and give up. How can a verifier possibly check every single piece of the computation, without looking at each piece of the computation individually? But it turns out that there is a clever solution.

Polynomials

Polynomials are a special class of algebraic expressions of the form:

- $(x + 5)$
- (x^4)
- $(x^3 + 3x^2 + 3x + 1)$
- $(628x^{271} + 318x^{270} + 530x^{269} + ... + 69x + 381)$

i.e. they are a sum of any (finite!) number of terms of the form $(c x^k)$.

There are many things that are fascinating about polynomials. But here we are going to zoom in on a particular one: **polynomials are a single mathematical object that can contain an unbounded amount of information** (think of them as a list of integers and this is obvious). The fourth example above contained 816 digits of [tau](#), and one can easily imagine a polynomial that contains far more.

Furthermore, **a single equation between polynomials can represent an unbounded number of equations between numbers**. For example, consider the equation $(A(x) + B(x) = C(x))$. If this equation is true, then it's also true that:

- $(A(0) + B(0) = C(0))$
- $(A(1) + B(1) = C(1))$
- $(A(2) + B(2) = C(2))$
- $(A(3) + B(3) = C(3))$

And so on for every possible coordinate. You can even construct polynomials to deliberately represent sets of numbers so you can check many equations all at once. For example, suppose that you wanted to check:

- 12 + 1 = 13
- 10 + 8 = 18
- 15 + 8 = 23
- 15 + 13 = 28

You can use a procedure called [Lagrange interpolation](#) to construct polynomials $A(x)$ that give (12, 10, 15, 15) as outputs at some specific set of coordinates (eg. (0, 1, 2, 3)), $B(x)$ the outputs (1, 8, 8, 13) on those same coordinates, and so forth. In fact, here are the polynomials:

- $A(x) = -2x^3 + \frac{19}{2}x^2 - \frac{19}{2}x + 12$
- $B(x) = 2x^3 - \frac{19}{2}x^2 + \frac{29}{2}x + 1$
- $C(x) = 5x + 13$

Checking the equation $A(x) + B(x) = C(x)$ with these polynomials checks all four above equations at the same time.

Comparing a polynomial to itself

You can even check relationships between a large number of *adjacent evaluations of the same polynomial* using a simple polynomial equation. This is slightly more advanced. Suppose that you want to check that, for a given polynomial F, $F(x+2) = F(x) + F(x+1)$ within the integer range $\{0, 1 ... 98\}$ (so if you *also* check $F(0) = F(1) = 1$, then $F(100)$ would be the 100th [Fibonacci](#) number).

As polynomials, $F(x+2) - F(x+1) - F(x)$ would not be exactly zero, as it could give arbitrary answers *outside* the range $x = \{0, 1 ... 98\}$. But we can do something clever. In general, there is a rule that if a polynomial P is zero across some set $S=\{x_1, x_2 ... x_n\}$ then it can be expressed as $P(x) = Z(x) * H(x)$, where $Z(x) =$ $((x - x_1) * (x - x_2) * ... * (x - x_n))$ and $H(x)$ is also a polynomial. In other words, **any polynomial that equals zero across some set is a (polynomial) multiple of the simplest (lowest-degree) polynomial that equals zero across that same set**.

Why is this the case? It is a nice corollary of polynomial long division: [the factor theorem](#). We know that, when dividing $P(x)$ by $Z(x)$, we will get a quotient $Q(x)$ and a remainer $R(x)$ which satisfy $P(x) = Z(x) * Q(x) + R(x)$, where the degree of the remainder $R(x)$ is strictly less than that of $Z(x)$. Since we know that P is zero on all of S, it means that R has to be zero on all of S as well. So we can simply compute $R(x)$ via polynomial interpolation, since it's a polynomial of degree at most $n-1$ and we know n values (the zeroes at S). Interpolating a polynomial with all zeroes gives the zero polynomial, thus $R(x) = 0$ and $H(x)= Q(x)$.

Going back to our example, if we have a polynomial F that encodes Fibonacci numbers (so $F(x+2) = F(x) + F(x+1)$ across $x = \{0, 1 ... 98\}$), then I can convince you that F *actually satisfies this condition* by proving that the polynomial $P(x) =$ $F(x+2) - F(x+1) - F(x)$ is zero over that range, by giving you the quotient:

$H(x) = \frac{F(x+2) - F(x+1) - F(x)}{Z(x)}$

Where $Z(x) = (x - 0) * (x - 1) * ... * (x - 98)$.

You can calculate $Z(x)$ yourself (ideally you would have it precomputed), check the equation, and if the check passes then $F(x)$ satisfies the condition!

Now, step back and notice what we did here. We converted a 100-step-long computation (computing the 100th Fibonacci number) into a single equation with polynomials. Of course, proving the N'th Fibonacci number is not an especially useful task, especially since Fibonacci numbers [have a closed form](#). But you can use exactly the same basic technique, just with some extra polynomials and some more complicated equations, to encode arbitrary computations with an arbitrarily large number of steps.

Now, if only there was a way to verify equations with polynomials that's much faster than checking each coefficient...

Polynomial commitments

And once again, it turns out that there is an answer: **polynomial commitments**. A polynomial commitment is best viewed as a special way to "hash" a polynomial, where the hash has the additional property that you can check equations between polynomials by checking equations between their hashes. Different polynomial commitment schemes have different properties in terms of exactly what kinds of equations you can check.

Here are some common examples of things you can do with various polynomial commitment schemes (we use $\text{com}(P)$ to mean "the commitment to the polynomial P"):

- **Add them**: given $\text{com}(P)$, $\text{com}(Q)$ and $\text{com}(R)$ check if $P + Q = R$
- **Multiply them**: given $\text{com}(P)$, $\text{com}(Q)$ and $\text{com}(R)$ check if $P * Q = R$
- **Evaluate at a point**: given $\text{com}(P)$, w, z and a supplemental proof (or "witness") Q, verify that $P(w) = z$

It's worth noting that these primitives can be constructed from each other. If you can add and multiply, then you can evaluate: to prove that $P(w) = z$, you can construct $Q(x) = \frac{P(x) - z}{x - w}$, and the verifier can check if $Q(x) * (x - w) + z \stackrel{?}{=} P(x)$. This works because if such a polynomial $Q(x)$ *exists*, then $P(x) - z = Q(x) * (x - w)$, which means that $P(x) - z$ equals zero at w (as $x - w$ equals zero at w) and so $P(x)$ equals z at w.

And if you can evaluate, you can do all kinds of checks. This is because there is a [mathematical theorem](#) that says, approximately, that if some equation involving some polynomials holds true at a *randomly selected coordinate*, then it almost certainly holds true for the polynomials as a whole. So if all we have is a mechanism to prove evaluations, we can check eg. our equation $P(x + 2) - P(x + 1) - P(x) = Z(x) * H(x)$ using an interactive game:
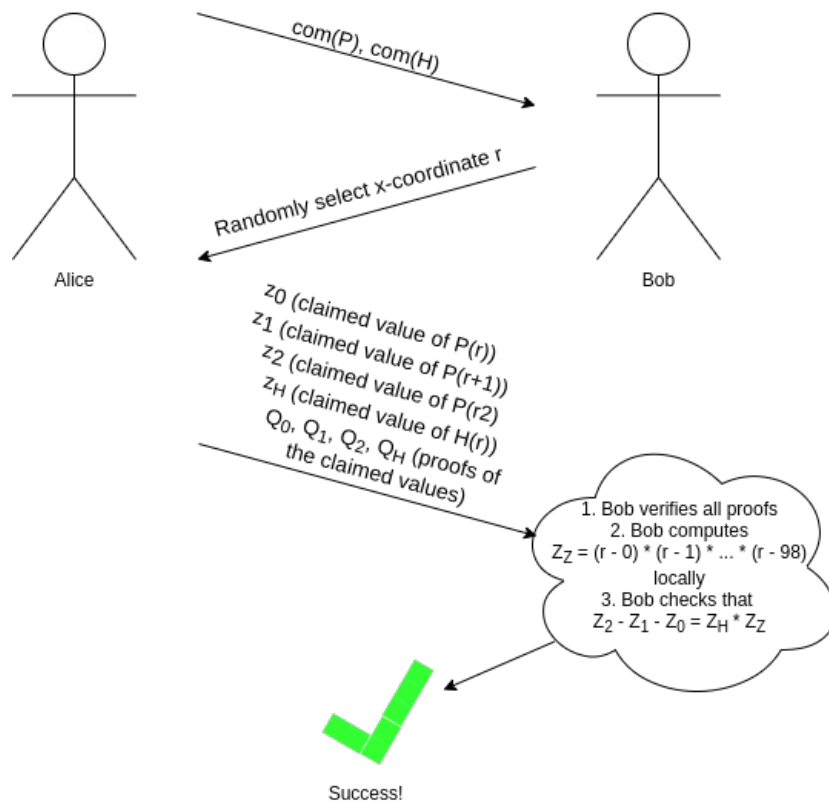
As I alluded to earlier, we can make this *non-interactive* using the **Fiat-Shamir heuristic**: the prover can compute r themselves by setting `r = hash(com(P), com(H))` (where `hash` is any cryptographic hash function; it does not need any special properties). The prover cannot "cheat" by picking P and H that "fit" at that particular r but not elsewhere, because they do not know r at the time that they are picking P and H!

A quick recap so far

- ZK-SNARKs are hard because the verifier needs to somehow check millions of steps in a computation, without doing a piece of work to check each individual step directly (as that would take too long).
- We get around this by encoding the computation into polynomials.
- A single polynomial can contain an unboundedly large amount of information, and a single polynomial expression (eg. $P(x+2) - P(x+1) - P(x) = Z(x) * H(x)$) can "stand in" for an unboundedly large number of equations between numbers.
- If you can verify the equation with polynomials, you are implicitly verifying all of the number

equations (replace x with any actual x-coordinate) simultaneously.
- We use a special type of "hash" of a polynomial, called a *polynomial commitment*, to allow us to actually verify the equation between polynomials in a very short amount of time, even if the underlying polynomials are very large.

So, how do these fancy polynomial hashes work?

There are three major schemes that are widely used at the moment: **bulletproofs, Kate and FRI**.

- Here is a description of Kate commitments by Dankrad Feist: https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html
- Here is a description of bulletproofs by the curve25519-dalek team: https://doc-internal.dalek.rs/bulletproofs/notes/inner_product_proof/index.html, and here is an explanation-in-pictures by myself: https://twitter.com/VitalikButerin/status/1371844878968176647
- Here is a description of FRI by... myself: https://vitalik.ca/general/2017/11/22/starks_part_2.html

Whoa, whoa, take it easy. Try to explain one of them simply, without shipping me off to even more scary links

To be honest, they're not *that* simple. There's a reason why all this math did not really take off until 2015 or so.

Please?

In my opinion, the easiest one to understand fully is FRI (Kate is easier if you're willing to accept elliptic curve pairings as a "black box", but pairings are *really* complicated, so altogether I find FRI simpler).

Here is how a simplified version of FRI works (the real protocol has many tricks and optimizations that are missing here for simplicity). Suppose that you have a polynomial P with degree $< n$. The commitment to P is a Merkle root of a set of evaluations to P at some set of pre-selected coordinates (eg. $\{0, 1 .... 8n-1\}$, though this is not the most efficient choice). Now, we need to add something extra to prove that this set of evaluations actually is a degree $< n$ polynomial.

Let Q be the polynomial only containing the even coefficients of P, and R be the polynomial only containing the odd coefficients of P. So if $P(x) = x^4 + 4x^3 + 6x^2 + 4x + 1$, then $Q(x) = x^2 + 6x + 1$ and $R(x) = 4x + 4$ (note that the degrees of the coefficients get "collapsed down" to the range $[0...\frac{n}{2}]$).

Notice that $P(x) = Q(x^2) + x * R(x^2)$ (if this isn't immediately obvious to you, stop and think and look at the example above until it is).

We ask the prover to provide Merkle roots for $Q(x)$ and $R(x)$. We then generate a random number r and ask the prover to provide a "random linear combination" $S(x) = Q(x) + r * R(x)$.

We pseudorandomly sample a large set of indices (using the already-provided Merkle roots as the seed for the randomness as before), and ask the prover to provide the Merkle branches for P, Q, R and S at these indices. At each of these provided coordinates, we check that:

- $P(x)$ *actually does equal* $Q(x^2) + x * R(x^2)$
- $S(x)$ *actually does equal* $Q(x) + r * R(x)$

If we do enough checks, then we can be convinced that the "expected" values of $S(x)$ are different from the "provided" values in at most, say, 1% of cases.

Notice that Q and R both have degree $< \frac{n}{2}$. Because S is a linear combination of Q and R, S *also* has degree $< \frac{n}{2}$. And this works in reverse: if we can prove S has degree $< \frac{n}{2}$, then the fact that it's a randomly chosen combination prevents the prover from choosing malicious Q and R with hidden high-degree coefficients that "cancel out", so Q and R must both be degree $< \frac{n}{2}$, and because $P(x) = Q(x^2) + x * R(x^2)$, we know that P must have degree $< n$.

From here, we simply repeat the game with S, progressively "reducing" the polynomial we care about to a lower and lower degree, until it's at a sufficiently low degree that we can check it directly.
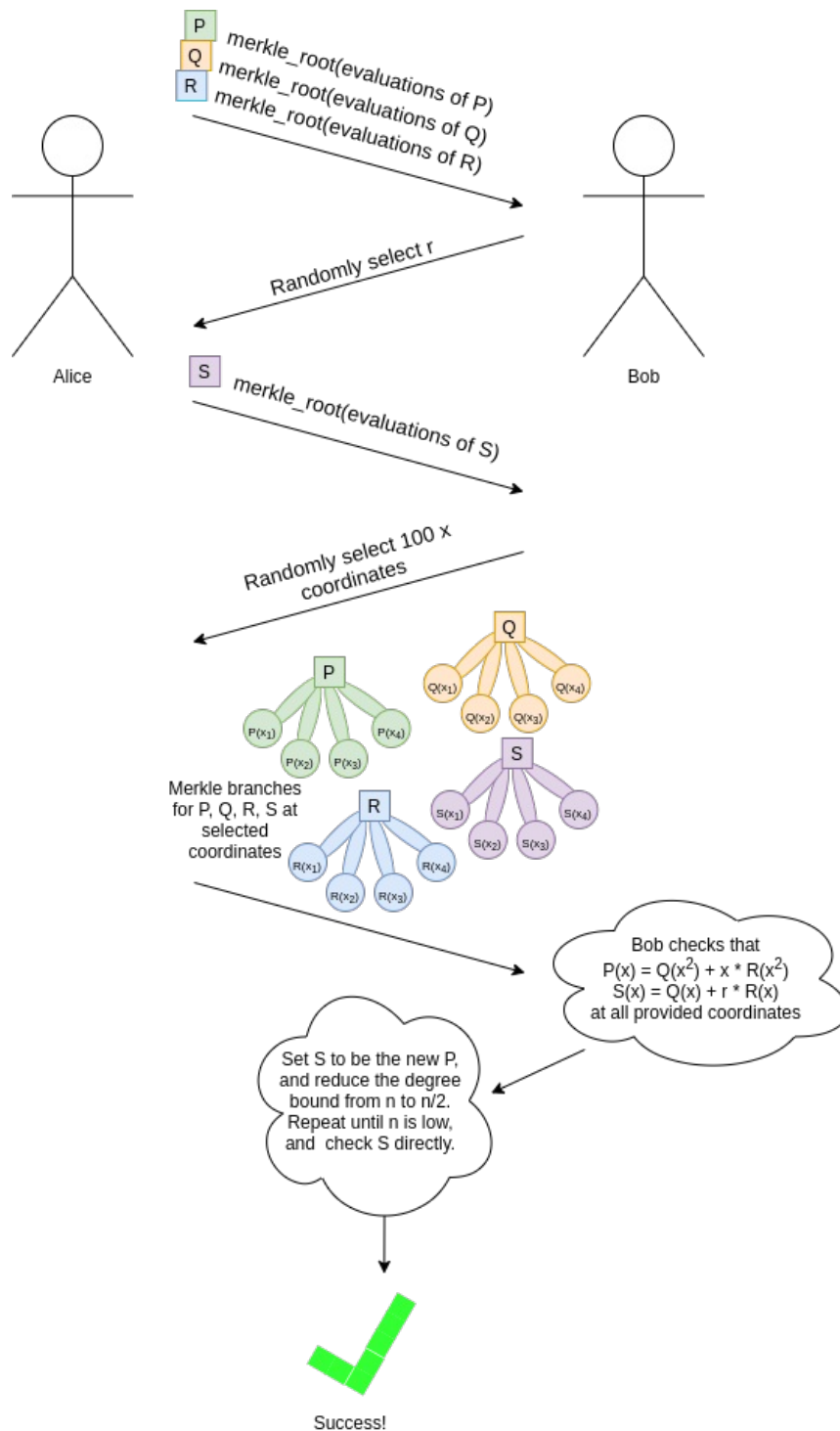
As in the previous examples, "Bob" here is an abstraction, useful for cryptographers to mentally reason about the protocol. In reality, Alice is generating the entire proof herself, and to prevent her from cheating we use Fiat-Shamir: we choose each randomly samples coordinate or r value based on the hash of the data generated in the proof up until that point.

A full "FRI commitment" to P (in this simplified protocol) would consist of:

1. The Merkle root of evaluations of P
2. The Merkle roots of evaluations of Q, R, S_1
3. The randomly selected branches of P, Q, R, S_1 to check S_1 is correctly "reduced from" P
4. The Merkle roots and randomly selected branches just as in steps (2) and (3) for successively lower-degree reductions S_2 reduced from S_1, S_3 reduced from S_2, all the way down to a low-degree S_k (this gets repeated $\approx \log_2(n)$ times in total)

5. The full Merkle tree of the evaluations of S_k (so we can check it directly)

Each step in the process can introduce a bit of "error", but if you add enough checks, then the total error will be low enough that you can prove that $P(x)$ equals a degree $< n$ polynomial in at least, say, 80% of positions. And this is sufficient for our use cases. If you want to cheat in a zk-SNARK, you would need to make a polynomial commitment for a fractional expression (eg. to "prove" the false claim that $x^2 + 2x + 3$ evaluated at 4 equals 5, you would need to provide a polynomial commitment for $\frac{x^2 + 2x + 3 - 5}{x - 4} = x + 6 + \frac{22}{x - 4}$). The set of evaluations for such a fractional expression would *differ* from the evaluations for any real degree $< n$ polynomial in so many positions that any attempt to make a FRI commitment to them would fail at some step.

Also, you can check carefully that the total number and size of the objects in the FRI commitment is logarithmic in the degree, so for large polynomials, the commitment really is much smaller than the polynomial itself.

To check equations between different polynomial commitments of this type (eg. check $A(x) + B(x) = C(x)$ given FRI commitments to A, B and C), simply randomly select many indices, ask the prover for Merkle branches at each of those indices for each polynomial, and verify that the equation actually holds true at each of those positions.

The above description is a highly inefficient protocol; there is a whole host of algebraic tricks that can increase its efficiency by a factor of something like a hundred, and you need these tricks if you want a protocol that is actually viable for, say, use inside a blockchain transaction. In particular, for example, Q and R are not actually necessary, because if you choose your evaluation points very cleverly, you can reconstruct the evaluations of Q and R that you need directly from evaluations of P. But the above description should be enough to convince you that a polynomial commitment is fundamentally possible.

Finite fields

In the descriptions above, there was a hidden assumption: that each individual "evaluation" of a polynomial was small. But when we are dealing with polynomials that are big, this is clearly not true. If we take our example from above, $628x^{271} + 318x^{270} + 530x^{269} + ... + 69x + 381$, that encodes 816 digits of tau, and evaluate it at $x=1000$, you get.... an 816-digit number containing all of those digits of tau. And so there is one more thing that we need to add. In a real implementation, all of the arithmetic that we are doing here would not be done using "regular" arithmetic over real numbers. Instead, it would be done using *modular arithmetic*.

We redefine all of our arithmetic operations as follows. We pick some prime "modulus" p. The % operator means "take the remainder of": $15\ \%\ 7 = 1$, $53\ \%\ 10 = 3$, etc (note that the answer is always non-negative, so for example $-1\ \%\ 10 = 9$). We redefine

$(x + y \Rightarrow (x + y)$ % p

$(x * y \Rightarrow (x * y)$ % p

$(x^y \Rightarrow (x^y)$ % p

$(x - y \Rightarrow (x - y)$ % p

$(x / y \Rightarrow (x * y ^{p-2})$ % p

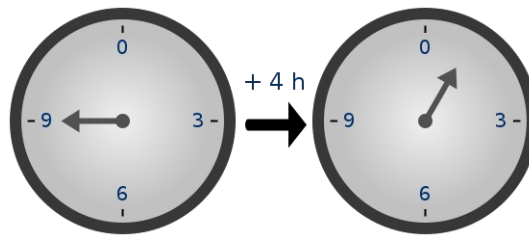The above rules are all self-consistent. For example, if $p = 7$, then:

* $5 + 3 = 1$ (as 8 % $7 = 1$)
* $1 - 3 = 5$ (as -2 % $7 = 5$)
* $2 \cdot 5 = 3$
* $3 / 5 = 2$ (as ($3 \cdot 5^5$) % $7 = 9375$) % $7 = 2$)

More complex identities such as the distributive law also hold: $(2 + 4) \cdot 3$ and $2 \cdot 3 + 4 \cdot 3$ both evaluate to 4. Even formulas like $(a^2 - b^2)$ = $(a - b) \cdot (a + b)$ are still true in this new kind of arithmetic.

Division is the hardest part; we can't use regular division because we want the values to always remain integers, and regular division often gives non-integer results (as in the case of $3/5$). We get around this problem using [Fermat's little theorem](#), which states that for any nonzero $x < p$, it holds that x^{p-1} % $p = 1$. This implies that x^{p-2} gives a number which, if multiplied

by x one more time, gives 1, and so we can say that x^{p-2} (which is an integer) equals $\frac{1}{x}$. A somewhat more complicated but faster way to evaluate this modular division operator is the [extended Euclidean algorithm](), implemented in python [here]().

Because of how the numbers "wrap around", modular arithmetic is sometimes called "clock math"

With modular math we've created an entirely new system of arithmetic, and it's self-consistent in all the same ways traditional arithmetic is self-consistent. Hence, we can talk about all of the same kinds of structures over this field, including polynomials, that we talk about in "regular math". Cryptographers love working in modular math (or, more generally, "finite fields") because there is a bound on the size of a number that can arise as a result of any modular math calculation - no matter what you do, the values will not "escape" the set $\{0, 1, 2 ... p-1\}$. Even evaluating a degree-1-million polynomial in a finite field will never give an answer outside that set.

What's a slightly more useful example of a computation being converted into a set of polynomial equations?

Let's say we want to prove that, for some polynomial P, $0 \le P(n) < 2^{64}$, without revealing the exact value of $P(n)$. This is a common use case in blockchain transactions, where you want to prove that a transaction leaves a balance non-negative without revealing what that balance is.

We can construct a proof for this with the following polynomial equations (assuming for simplicity $n = 64$):

- $P(0) = 0$
- $P(x+1) = P(x) * 2 + R(x)$ across the range $\{0...63\}$
- $R(x) \in \{0,1\}$ across the range $\{0...63\}$

The latter two statements can be restated as "pure" polynomial equations as follows (in this context $Z(x) = (x - 0) * (x - 1) * ... * (x - 63)$):

- $P(x+1) - P(x) * 2 - R(x) = Z(x) * H\_1(x)$
- $R(x) * (1 - R(x)) = Z(x) * H\_2(x)$ (notice the clever trick: $y * (1-y) = 0$ if and only if $y \in \{0, 1\}$)

The idea is that successive evaluations of $P(i)$ build up the number bit-by-bit: if $P(4) = 13$, then the sequence of evaluations going up to that point would be: $\{0, 1, 3, 6, 13\}$. In binary, 1 is 1, 3 is 11, 6 is 110, 13 is 1101; notice how $P(x+1) = P(x) * 2 + R(x)$ keeps adding one bit to the end as long as $R(x)$ is zero or one. Any number within the range $0 \le x < 2^{64}$ can be built up over 64 steps in this way, any number outside that range cannot.

Privacy

But there is a problem: how do we know that the commitments to $P(x)$ and $R(x)$ don't "leak" information that allows us to uncover the exact value of $P(64)$, which we are trying to keep hidden?

There is some good news: **these proofs are small proofs that can make statements about a large amount of data and computation. So in general, the proof will very often simply *not be big enough* to leak more than a little bit of information**. But can we go from "only a little bit" to "zero"? Fortunately, we can.

Here, one fairly general trick is to add some "fudge factors" into the polynomials. When we choose P, add a small multiple of $Z(x)$ into the polynomial (that is, set $P'(x) = P(x) + Z(x) * E(x)$ for some random $E(x)$). This does not affect the correctness of the statement (in fact, P' evaluates

to the same values as \(P\) on the coordinates that "the computation is happening in", so it's still a valid transcript), but it can add enough extra "noise" into the commitments to make any remaining information unrecoverable. Additionally, in the case of FRI, it's important to not sample random points that are within the domain that computation is happening in (in this case \(\{0...64\}\)).

Can we have one more recap, please??

- The three most prominent types of polynomial commitments are FRI, Kate and bulletproofs.
- Kate is the simplest conceptually but depends on the really complicated "black box" of elliptic curve pairings.
- FRI is cool because it relies only on hashes; it works by successively reducing a polynomial to a lower and lower-degree polynomial and doing random sample checks with Merkle branches to prove equivalence at each step.
- To prevent the size of individual numbers from blowing up, instead of doing arithmetic and polynomials *over the integers*, we do everything *over a finite field* (usually integers modulo some prime `p`)
- Polynomial commitments lend themselves naturally to privacy preservation because the proof is already much smaller than the polynomial, so a polynomial commitment can't reveal more than a little bit of the information in the polynomial anyway. But we can add some randomness to the polynomials we're committing to to reduce the information revealed from "a little bit" to "zero".

What research questions are still being worked on?

- **Optimizing FRI**: there are already quite a few optimizations involving carefully selected evaluation domains, "DEEP-FRI", and a whole host of other tricks to make FRI more efficient. Starkware and others are working on this.
- **Better ways to encode computation into polynomials**: figuring out the most efficient way to encode complicated computations involving hash functions, memory access and other features into polynomial equations is still a challenge. There has been great progress on this (eg. see PLOOKUP), but we still need more, especially if we want to encode general-purpose virtual machine execution into polynomials.
- **Incrementally verifiable computation**: it would be nice to be able to efficiently keep "extending" a proof *while* a computation continues. This is valuable in the "single-prover" case, but also in the "multi-prover" case, particularly a blockchain where a different participant creates each block. See Halo for some recent work on this.

I wanna learn more!

My materials

- STARKs: part 1, part 2, part 3
- Specific protocols for encoding computation into polynomials: PLONK
- Some key mathematical optimizations I didn't talk about here: Fast Fourier transforms

Other people's materials

- Starkware's online course
- Dankrad Feist on Kate commitments
- Bulletproofs

Why we need wide adoption of social recovery wallets

2021 Jan 11

Special thanks to Itamar Lesuisse from Argent and Daniel Wang from Loopring for feedback.

One of the great challenges with making cryptocurrency and blockchain applications usable for average users is security: how do we prevent users' funds from being lost or stolen? Losses and thefts are a serious issue, often costing innocent blockchain users thousands of dollars or even in some cases the majority of their entire net worth.

There have been many solutions proposed over the years: paper wallets, hardware wallets, and my own one-time favorite: multisig wallets. And indeed they have led to significant improvements in security. However, these solutions have all suffered from various defects - sometimes providing far less extra protection against theft and loss than is actually needed, sometimes being cumbersome and difficult to use leading to very low adoption, and sometimes both. But recently, there is an emerging better alternative: a newer type of smart contract wallet called a *social recovery wallet*. These wallets can potentially provide a high level of security and much better usability than previous options, but there is still a way to go before they can be easily and widely deployed. **This post will go through what social recovery wallets are, why they matter, and how we can and should move toward much broader adoption of them throughout the ecosystem.**

Wallet security is a really big problem

Wallet security issues have been a thorn in the side of the blockchain ecosystem almost since the beginning. Cryptocurrency losses and thefts were rampant even back in 2011 when Bitcoin was almost the only cryptocurrency out there; indeed, in my pre-Ethereum role as a cofounder and writer of Bitcoin Magazine, I wrote an entire article detailing the horrors of hacks and losses and thefts that were already happening at the time.

Here is one sample:

> Last night around 9PM PDT, I clicked a link to go to CoinChat[.]freetzi[.]com – and I was prompted to run java. I did (thinking this was a legitimate chatoom), and nothing happened. I closed the window and thought nothing of it. I opened my bitcoin-qt wallet approx 14 minutes later, and saw a transaction that I did NOT approve go to wallet 1Es3QVvKN1qA2p6me7jLCVMZpQXVXWPNTC for almost my entire wallet...

This person's losses were 2.07 BTC, worth $300 at the time, and over $70000 today. Here's another one:

> In June 2011, the Bitcointalk member "allinvain" lost 25,000 BTC (worth $500,000 at the time) after an unknown intruder somehow gained direct access to his computer. The attacker was able to access allinvain's wallet.dat file, and quickly empty out the wallet – either by sending a transaction from allinvain's computer itself, or by simply uploading the wallet.dat file and emptying it on his own machine.

In present-day value, that's a loss of *nearly one billion dollars*. But theft is not the only concern; there are also losses from losing one's private keys. Here's Stefan Thomas:

> Bitcoin developer Stefan Thomas had three backups of his wallet – an encrypted USB stick, a Dropbox account and a Virtualbox virtual machine. However, he managed to erase two of them and forget the password to the third, forever losing access to 7,000 BTC (worth $125,000 at the time). Thomas's reaction: "[I'm] pretty dedicated to creating better clients since then."

One analysis of the Bitcoin ecosystem suggests that 1500 BTC may be lost every day - over ten times more than what Bitcoin users spend on transaction fees, and over the years adding up to as much as 20% of the total supply. The stories and the numbers alike point to the same inescapable truth: **the importance of the wallet security problem is great, and it should not be underestimated**.

It's easy to see the social and psychological reasons why wallet security is easy to underestimate: people naturally worry about appearing uncareful or dumb in front of an always judgemental public, and so many keep their experiences with their funds getting hacked to themselves. Loss of funds is even worse, as there is a pervasive (though in my opinion very incorrect) feeling that "there is no one to blame but yourself". **But the reality is that *the whole point of digital technology, blockchains included, is to make it easier for humans to engage in very complicated tasks without having to exert extreme mental effort or live in constant fear of making mistakes.*** An ecosystem whose only answer to losses and thefts is a combination of 12-step tutorials, not-very-secure half-measures and the not-so-occasional semi-sarcastic "sorry for your loss" is going to have a hard time getting broad adoption.

So solutions that reduce the quantity of losses and thefts taking place, without requiring all cryptocurrency users to turn personal security into a full-time hobby, are highly valuable for the industry.

Hardware wallets alone are not good enough

Hardware wallets are often touted as the best-in-class technology for cryptocurrency funds management. A hardware wallet is a specialized hardware device which can be connected to your computer or phone (eg. through USB), and which contains a specialized chip that can only generate private keys and sign transactions. A transaction would be initiated on your computer or phone, must be confirmed on the hardware wallet before it can be sent. The private key stays on your hardware wallet, so an attacker that hacks into your computer or phone could *not* drain the funds.

Hardware wallets are a significant improvement, and they certainly would have protected the Java chatroom victim, but they are not perfect. I see two main problems with hardware wallets:

- **Supply chain attacks**: if you buy a hardware wallet, you are trusting a number of actors that were involved in producing it - the company that designed the wallet, the factory that produced it, and everyone involved in shipping it who could have replaced it with a fake. Hardware wallets are potentially a magnet for such attacks: the ratio of funds stolen to number of devices compromised is very high. To their credit, hardware wallet manufacturers such as Ledger have put in many safeguards to protect against these risks, but some risks still remain. A hardware device fundamentally cannot be audited the same way a piece of open source software can.
- **Still a single point of failure**: if someone steals your hardware wallet right after they stand behind your shoulder and catch you typing in the PIN, they can steal your funds. If you lose your hardware wallet, then you lose your funds - unless the hardware wallet generates and outputs a backup at setup time, but as we will see those have problems of their own...

Mnemonic phrases are not good enough

Many wallets, hardware and software alike, have a setup procedure during which they output a *mnemonic phrase*, which is a human-readable 12 to 24-word encoding of the wallet's root private key. A mnemonic phrase looks like this:

```
vote    dance   type     subject valley  fall    usage   silk
essay   lunch   endorse  lunar   obvious race    ribbon  key
already arrow   enable   drama   keen    survey  lesson  cruel
```

If you lose your wallet but you have the mnemonic phrase, you can input the phrase when setting up a new wallet to recover your account, as the mnemonic phrase contains the root key from which all of your other keys can be generated.

Mnemonic phrases are good for protecting against loss, but they do nothing against theft. Even worse, they add a *new* vector for theft: if you have the standard hardware wallet + mnemonic backup combo, then someone stealing *either* your hardware wallet + PIN *or* your mnemonic backup can steal your funds. Furthermore, maintaining a mnemonic phrase and not accidentally throwing it away is itself a non-trivial mental effort.

The problems with theft can be alleviated if you split the phrase in half and give half to your friend, but (i) almost no one actually promotes this, (ii) there are security issues, as if the phrase is short (128 bits) then a sophisticated and motivated attacker who steals one piece may be able to brute-force through all 2^{64} possible combinations to find the other, and (iii) it increases the mental overhead even further.

So what do we need?

What we need is a wallet design which satisfies three key criteria:

- **No single point of failure**: there is no single thing (and ideally, no collection of things which travel together) which, if stolen, can give an attacker access to your funds, or if lost, can deny you access to your funds.
- **Low mental overhead**: as much as possible, it should not require users to learn strange new habits or exert mental effort to always remember to follow certain patterns of behavior.
- **Maximum ease of transacting**: most normal activities should not require much more effort than they do in regular wallets (eg. Status, Metamask...)

Multisig is good!

The best-in-class technology for solving these problems [back in 2013](#) was multisig. You could have a wallet that has three keys, where any two of them are needed to send a transaction.
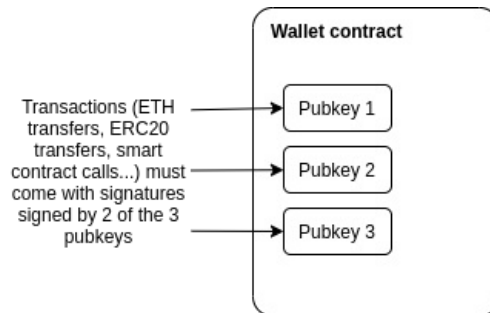
This technology was originally developed within the Bitcoin ecosystem, but excellent multisig wallets (eg. see [Gnosis Safe](#)) now exist for Ethereum too. Multisig wallets have been highly successful within organizations: the Ethereum Foundation uses a 4-of-7 multisig wallet to store [its funds](#), as do many other orgs in the Ethereum ecosystem.

For a multisig wallet to hold the funds for an *individual*, the main challenge is: who holds the funds, and how are transactions approved? The most common formula is some variant of "two easily accessible, but separate, keys, held by you (eg. laptop and phone) and a third more secure but less accessible a backup, held offline or by a friend or institution".

This is reasonably secure: there is no single device that can be lost or stolen that would lead to you losing access to your funds. But the security is far from perfect: if you can steal someone's laptop, it's often not that hard to steal their phone as well. The usability is also a challenge, as every transaction now requires two confirmations with two devices.

Social recovery is better

This gets us to my preferred method for securing a wallet: social recovery. A social recovery system works as follows:

1. There is a single "signing key" that can be used to approve transactions
2. There is a set of at least 3 (or a much higher number) of "guardians", of which a majority can cooperate to change the signing key of the account.

The signing key has the ability to add or remove guardians, though only after a delay (often 1-3 days).
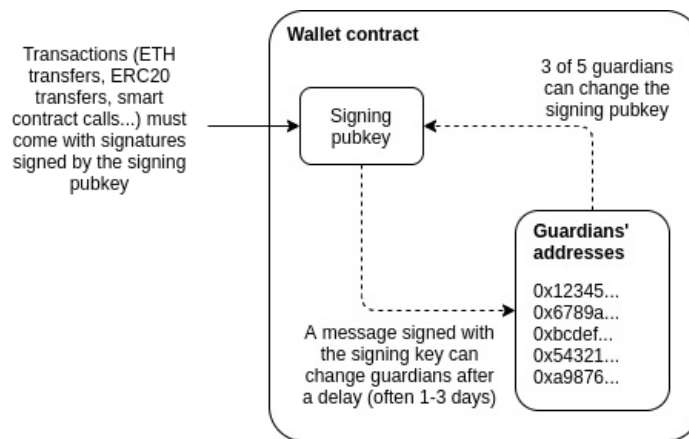
Under all normal circumstances, the user can simply use their social recovery wallet like a regular wallet, signing messages with their signing key so that each transaction signed can fly off with a single confirmation click much like it would in a "traditional" wallet like Metamask.

If a user *loses* their signing key, that is when the social recovery functionality would kick in. The user can simply reach out to their guardians and ask them to sign a special transaction to change the signing pubkey registered in the wallet contract to a new one. This is easy: they can simply go to a webpage such as security.loopring.io, sign in, see a recovery request and sign it. About as easy for each guardian as making a Uniswap trade.

There are many possible choices for whom to select as a guardian. The three most common choices are:

* Other devices (or paper mnemonics) owned by the wallet holder themselves
* Friends and family members
* Institutions, which would sign a recovery message if they get a confirmation of your phone number or email or perhaps in high value cases verify you personally by video call

Guardians are easy to add: you can add a guardian simply by typing in their ENS name or ETH address, though most social recovery wallets will require the guardian to sign a transaction in the recovery webpage to agree to be added. In any sanely designed social recovery wallet, the guardian does NOT need to download and use the same wallet; they can simply use their existing Ethereum wallet, whichever type of wallet it is. Given the high convenience of adding guardians, if you are lucky enough that your social circles are already made up of Ethereum users, I personally favor high guardian counts (ideally 7+) for increased security. If you already have a wallet, there is no ongoing mental effort required to be a guardian: any recovery operations that you do would be done through your existing wallet. If you not know many other active Ethereum users, then a smaller number of guardians that you trust to be technically competent is best.

To reduce the risk of attacks on guardians and collusion, **your guardians do not have to be publicly known: in fact, they do not need to know *each other's* identities**. This can be accomplished in two ways. First, instead of the guardians' addresses being stored directly on chain, a hash of the list of addresses can be stored on chain, and the wallet owner would only need to publish the full list at recovery time. Second, each guardian can be asked to deterministically generate a new single-purpose address that they would use just for that particular recovery; they would not need to actually send any transactions with that address unless a recovery is actually required. To complement these technical protections, **it's recommended to choose a diverse collection of guardians from different social circles (including ideally one institutional guardian)**; these recommendations together would make it extremely difficult for the guardians to be attacked simultaneously or collude.

In the event that you die or are permanently incapacitated, it would be a socially agreed standard protocol that guardians can publicly announce themselves, so in that case they can find each other and recover your funds.

Social recovery wallets are not a betrayal, but rather an *expression*, of "crypto values"

One common response to suggestions to use any form of multisig, social recovery or otherwise, is the

idea that this solution goes back to "trusting people", and so is a betrayal of the values of the blockchain and cryptocurrency industry. While I understand why one may think this at first glance, I would argue that this criticism stems from a fundamental misunderstanding of what crypto should be about.

To me, the goal of crypto was never to remove the need for *all* trust. **Rather, the goal of crypto is to give people access to cryptographic and economic building blocks that give people more *choice* in whom to trust, and furthermore allow people to build more *constrained* forms of trust**: giving someone the power to do some things on your behalf without giving them the power to do everything. Viewed in this way, **multisig and social recovery are a perfect expression of this principle**: each participant has *some* influence over the ability to accept or reject transactions, but no one can move funds unilaterally. This more complex logic allows for a setup far more secure than what would be possible if there had to be one person or key that unilaterally controlled the funds.

This fundamental idea, that human inputs should be wielded carefully but not thrown away outright, is powerful because it works well with the strengths and weaknesses of the human brain. The human brain is quite poorly suited for remembering passwords and tracking paper wallets, but it's an ASIC for keeping track of relationships with other people. This effect is even stronger for less technical users: they may have a harder time with wallets and passwords, but they are just as adept at social tasks like "choose 7 people who won't all gang up on me". If we *can* extract at least some information from human inputs into a mechanism, without those inputs turning into a vector for attack and exploitation, then we should figure out how. And social recovery is very robust: for a wallet with 7 guardians to be compromised, 4 of the 7 guardians would need to somehow discover each other and agree to steal the funds, without *any* of them tipping the owner off: certainly a much tougher challenge than attacking a [wallet protected purely by a single individuals](#).

How can social recovery protect against theft?

Social recovery as explained above deals with the risk that you *lose* your wallet. But there is still the risk that your signing key gets *stolen*: someone hacks into your computer, sneaks up behind you while you're already logged in and hits you over the head, or even just uses some user interface glitch to trick you into signing a transaction that you did not intend to sign.

We can extend social recovery to deal with such issues by adding a [vault](#). Every social recovery wallet can come with an automatically generated vault. Assets can be moved to the vault just by sending them to the vault's address, but they can be moved out of the vault only with a 1 week delay. During that delay, the signing key (or, by extension, the guardians) can cancel the transaction. If desired, the vault could also be programmed so that some limited financial operations (eg. Uniswap trades between some whitelisted tokens) can be done without delay.

Existing social recovery wallets

Currently, the two major wallets that have implemented social recovery are the [Argent wallet](#) and the [Loopring wallet](#):
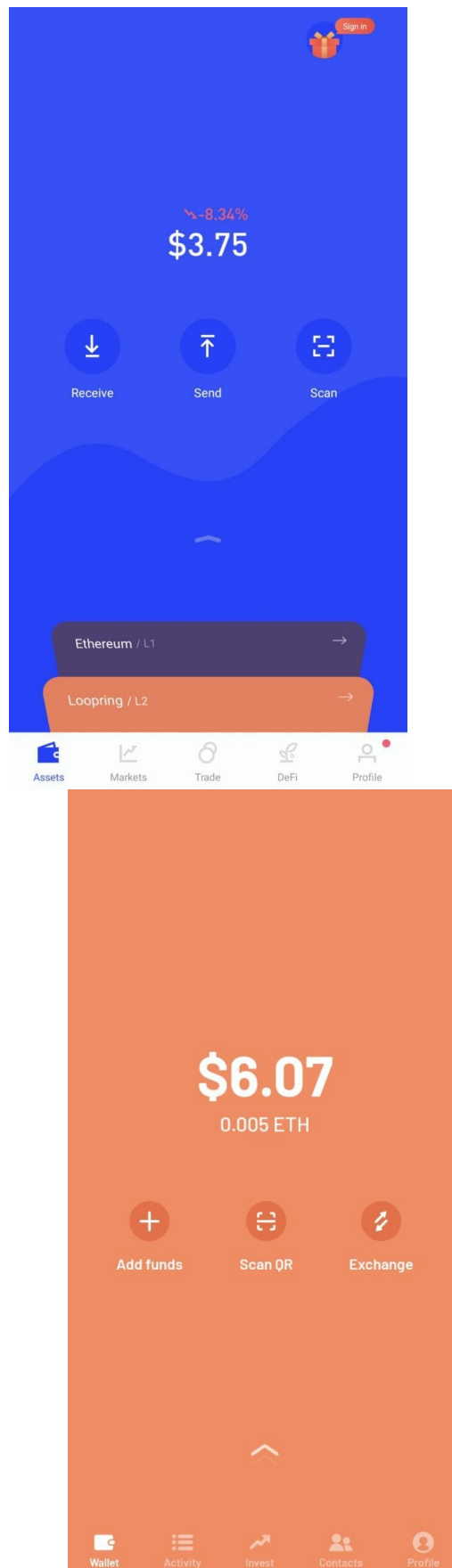
The Argent wallet is the first major, and still the most popular, "smart contract wallet" currently in use, and social recovery is one of its main selling points. The Argent wallet includes an interface by which guardians can be added and removed:
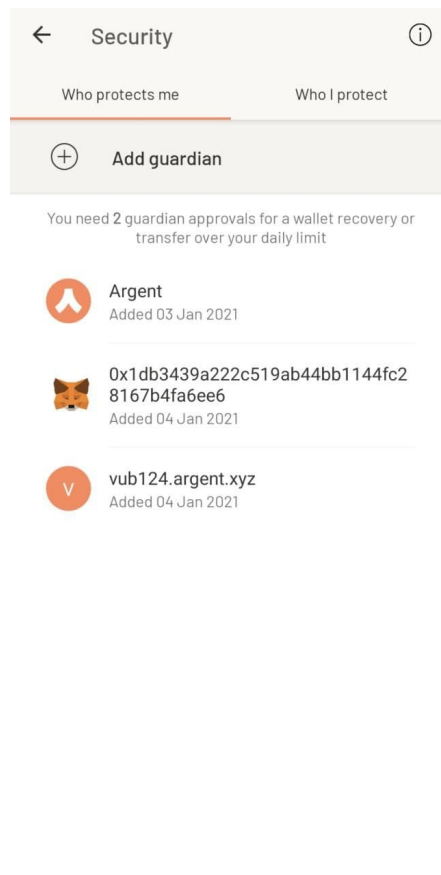
To protect against theft, the wallet has a daily limit: transactions up to that amount are instant but transactions above that amount require guardians to approve to finalize the withdrawal.

The Loopring wallet is most known for being built by the developers of (and of course including support for) the Loopring protocol, a ZK rollup for payments and decentralized exchange. But the Loopring wallet also has a social recovery feature, which works very similarly to that in Argent. In both cases, the wallet companies provide one guardian for free, which relies on a confirmation code sent by mobile phone to authenticate you. For the other guardians, you can add either other users of the same wallet, or any Ethereum user by providing their Ethereum address.

The user experience in both cases is surprisingly smooth. There were two main challenges. First, the smoothness in both cases relies on a central "relayer" run by the wallet maker that re-publishes signed messages as transactions. Second, the fees are high. Fortunately, both of these problems are surmountable.

Migration to Layer 2 (rollups) can solve the remaining challenges

As mentioned above, **there are two key challenges: (i) the dependence on relayers to solve transactions, and (ii) high transaction fees**. The first challenge, dependence on relayers, is an increasingly common problem in Ethereum applications. The issue arises because there are two types of accounts in Ethereum: **externally owned accounts (EOAs)**, which are accounts controlled by a single private key, and **contracts**. In Ethereum, there is a rule that every transaction must start from an EOA; the original intention was that EOAs represent "users" and contracts represent "applications", and an application can only run if a user talks to the application. If we want wallets with more complex policies, like multisig and social recovery, we need to use contracts to represent users. But this poses a challenge: if your funds are in a contract, you need to have some *other* account that has ETH that can pay to start each transaction, and it needs quite a lot of ETH just in case transaction fees get really high.

Argent and Loopring get around this problem by personally running a "relayer". The relayer listens for off-chain digitally signed "messages" submitted by users, and wraps these messages in a transaction and publishes them to chain. But for the long run, this is a poor solution; it adds an extra point of centralization. If the relayer is down and a user really needs to send a transaction, they can always just send it from their own EOA, but it is nevertheless the case that a new tradeoff between

centralization and inconvenience is introduced. There are efforts to solve this problem and get convenience without centralization; the main two categories revolve around either making a [generalized decentralized relayer network](#) or modifying the Ethereum protocol itself to [allow transactions to begin from contracts](#). But neither of these solutions solve transaction fees, and in fact, they make the problem worse due to smart contracts' inherently greater complexity.

Fortunately, we *can* solve both of these problems at the same time, by looking toward a third solution: moving the ecosystem [onto layer 2 protocols](#) such as optimistic rollups and ZK rollups. Optimistic and ZK rollups can both be designed with account abstraction built in, circumventing any need for relayers. Existing wallet developers are already looking into rollups, but ultimately migrating to rollups en masse is an ecosystem-wide challenge.

An ecosystem-wide mass migration to rollups is as good an opportunity as any to reverse the Ethereum ecosystem's earlier mistakes and give multisig and smart contract wallets a much more central role in helping to secure users' funds. But this requires broader recognition that wallet security is a challenge, and that we have not gone nearly as far in trying to meet and challenge as we should. Multisig and social recovery need not be the end of the story; there may well be designs that work even better. But the simple reform of moving to rollups and making sure that these rollups treat smart contract wallets as *first class citizens* is an important step toward making that happen.

An Incomplete Guide to Rollups

2021 Jan 05

Rollups are all the rage in the Ethereum community, and are [poised to be the key scalability solution](#) for Ethereum for the foreseeable future. But what exactly is this technology, what can you expect from it and how will you be able to use it? This post will attempt to answer some of those key questions.

Background: what is layer-1 and layer-2 scaling?

There are two ways to scale a blockchain ecosystem. First, you can make the blockchain itself have a higher transaction capacity. The main challenge with this technique is that blockchains with "bigger blocks" are inherently more difficult to verify and likely to become more centralized. To avoid such risks, developers can either increase the efficiency of client software or, more sustainably, use techniques [such as sharding](#) to allow the work of building and verifying the chain to be split up across many nodes; the [effort known as "eth2"](#) is currently building this upgrade to Ethereum.

Second, you can change the way that you use the blockchain. Instead of putting *all* activity on the blockchain directly, users perform the bulk of their activity off-chain in a "layer 2" protocol. There is a smart contract on-chain, which only has two tasks: processing deposits and withdrawals, and verifying proofs that everything happening off-chain is following the rules. There are multiple ways to do these proofs, but they all share the property that verifying the proofs on-chain is much cheaper than doing the original computation off-chain.

State channels vs plasma vs rollups

The three major types of layer-2 scaling are [state channels](#), [Plasma](#) and rollups. They are three different paradigms, with different strengths and weaknesses, and at this point we are fairly confident that all layer-2 scaling falls into roughly these three categories (though naming controversies exist at the edges, eg. see ["validium"](#)).

How do channels work?

See also: https://www.jeffcoleman.ca/state-channels and [statechannels.org](#)

Imagine that Alice is offering an internet connection to Bob, in exchange for Bob paying her $0.001 per megabyte. Instead of making a transaction for each payment, Alice and Bob use the following layer-2 scheme.

First, Bob puts $1 (or some ETH or stablecoin equivalent) into a smart contract. To make his first payment to Alice, Bob signs a "ticket" (an off-chain message), that simply says "$0.001", and sends it to Alice. To make his second payment, Bob would sign another ticket that says "$0.002", and send it to Alice. And so on and so forth for as many payments as needed. When Alice and Bob are done transacting, Alice can publish the highest-value ticket to chain, wrapped in another signature from herself. The smart contract verifies Alice and Bob's signatures, pays Alice the amount on Bob's ticket and returns the rest to Bob. If Alice is unwilling to close the channel (due to malice or technical failure), Bob can initiate a withdrawal period (eg. 7 days); if Alice does not provide a ticket within that time, then Bob gets all his money back.

This technique is powerful: it can be adjusted to handle bidirectional payments, smart contract relationships (eg. Alice and Bob making a financial contract inside the channel), and composition (if Alice and Bob have an open channel and so do Bob and Charlie, Alice can trustlessly interact with Charlie). But there are limits to what channels can do. Channels cannot be used to send funds off-chain to people who are not yet participants. Channels cannot be used to represent objects that do not have a clear logical owner (eg. Uniswap). And channels, especially if used to do things more complex than simple recurring payments, require a large amount of capital to be locked up.

How does plasma work?

See also: the [original Plasma paper](#), and [Plasma Cash](#).

To deposit an asset, a user sends it to the smart contract managing the Plasma chain. The Plasma chain assigns that asset a new unique ID (eg. 537). Each Plasma chain has an *operator* (this could be a centralized actor, or a multisig, or something more complex like PoS or DPoS). Every interval (this could be 15 seconds, or an hour, or anything in between), the operator generates a "batch" consisting of all of the Plasma transactions they have received off-chain. They generate a Merkle tree, where at each index X in the tree, there is a transaction transferring asset ID X if such a transaction exists, and otherwise that leaf is zero. They

publish the Merkle root of this tree to chain. They also send the Merkle branch of each index X to the current owner of that asset. To withdraw an asset, a user publishes the Merkle branch of the most recent transaction sending the asset to them. The contract starts a challenge period, during which anyone can try to use other Merkle branches to invalidate the exit by proving that either (i) the sender did not own the asset at the time they sent it, or (ii) they sent the asset to someone else at some later point in time. If no one proves that the exit is fraudulent for (eg.) 7 days, the user can withdraw the asset.

Plasma provides stronger properties than channels: you can send assets to participants who were never part of the system, and the capital requirements are much lower. But it comes at a cost: channels require no data whatsoever to go on chain during "normal operation", but Plasma requires each chain to publish one hash at regular intervals. Additionally, Plasma transfers are not instant: you have to wait for the interval to end and for the block to be published.

Additionally, Plasma and channels share a key weakness in common: the game theory behind why they are secure relies on the idea that each object controlled by both systems has some logical "owner". If that owner does not care about their asset, then an "invalid" outcome involving that asset may result. This is okay for many applications, but it is a deal breaker for many others (eg. Uniswap). Even systems where the state of an object can be changed without the owner's consent (eg. account-based systems, where you can *increase* someone's balance without their consent) do not work well with Plasma. This all means that a large amount of "application-specific reasoning" is required in any realistic plasma or channels deployment, and it is not possible to make a plasma or channel system that just simulates the full ethereum environment (or "the EVM"). To get around this problem, we get to... rollups.

Rollups

*See also: [EthHub on optimistic rollups](#) and [ZK rollups](#).*

Plasma and channels are "full" layer 2 schemes, in that they try to move both data *and* computation off-chain. However, [fundamental game theory issues around data availability](#) means that it is impossible to safely do this for all applications. Plasma and channels get around this by relying on an explicit notion of owners, but this prevents them from being fully general. Rollups, on the other hand, are a "hybrid" layer 2 scheme. **Rollups move computation (and state storage) off-chain, but keep some data per transaction on-chain**. To improve efficiency, they use a whole host of fancy compression tricks to *replace data with computation* wherever possible. The result is a system where scalability is still limited by the data bandwidth of the underlying blockchain, but at a very favorable ratio: whereas an Ethereum base-layer ERC20 token transfer costs ~45000 gas, an ERC20 token transfer in a rollup takes up 16 bytes of on-chain space and costs under 300 gas.

The fact that data is on-chain is key (note: putting data "on IPFS" does *not* work, because IPFS does not provide *consensus* on whether or not any given piece of data is available; the data *must* go on a blockchain). Putting data on-chain and having consensus on that fact allows anyone to locally process all the operations in the rollup if they wish to, allowing them to detect fraud, initiate withdrawals, or personally start producing transaction batches. The lack of data availability issues means that a malicious or offline operator can do *even less* harm (eg. they cannot cause a 1 week delay), opening up a much larger design space for who has the right to publish batches and making rollups vastly easier to reason about. And most importantly, the lack of data availability issues means that there is no longer any need to map assets to owners, leading to the key reason why the Ethereum community is so much more excited about rollups than previous forms of layer 2 scaling: **rollups are fully general-purpose, and one can even run an EVM inside a rollup, allowing existing Ethereum applications to migrate to rollups with almost no need to write any new code**.

OK, so how exactly does a rollup work?

There is a smart contract on-chain which maintains a **state root**: the Merkle root of the state of the rollup (meaning, the account balances, contract code, etc, that are "inside" the rollup).
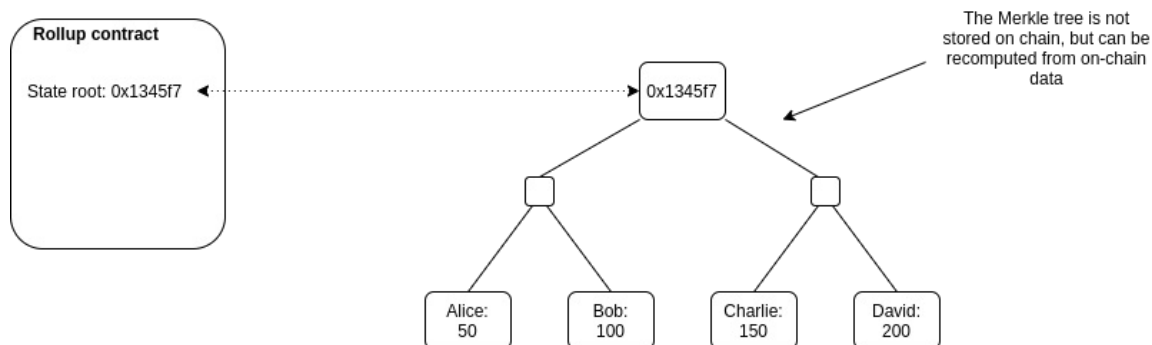
Anyone can publish a **batch**, a collection of transactions in a highly compressed form together with the previous state root and the new state root (the Merkle root *after* processing the transactions). The contract checks that the previous state root in the batch matches its current state root; if it does, it switches the state root to the new state root.
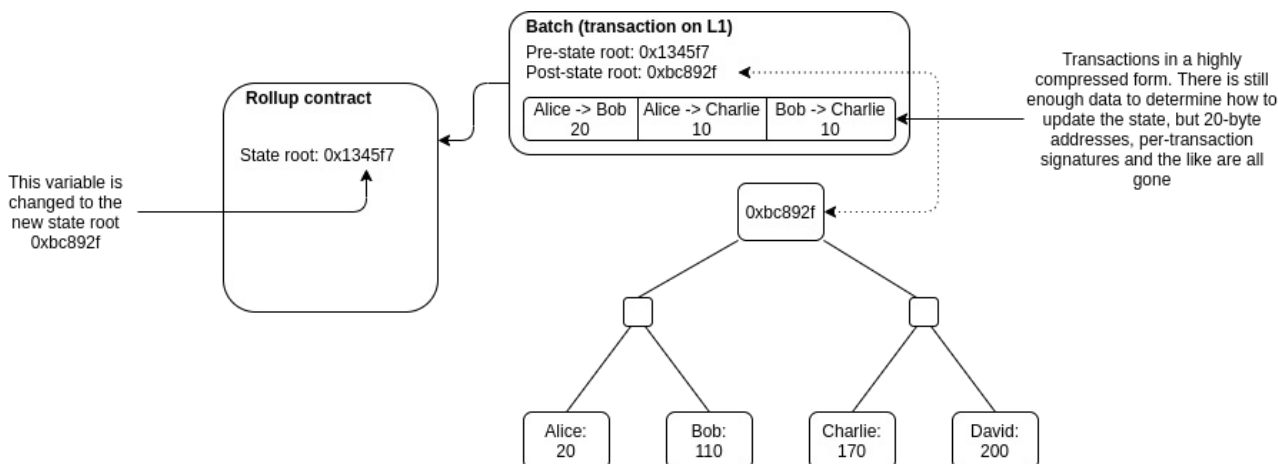
To support depositing and withdrawing, we add the ability to have transactions whose input or output is "outside" the rollup state. If a batch has inputs from the outside, the transaction submitting the batch needs to also transfer these assets to the rollup contract. If a batch has outputs to the outside, then upon processing the batch the smart contract initiates those withdrawals.

And that's it! Except for one major detail: **how to do know that the post-state roots in the batches are correct?** If someone can submit a batch with any post-state root with no consequences, they could just transfer all the coins inside the rollup to themselves. This question is key because there are two very different families of solutions to the problem, and these two families of solutions lead to the two flavors of rollups.

Optimistic rollups vs ZK rollups

The two types of rollups are:

1. **Optimistic rollups**, which use **fraud proofs**: the rollup contract keeps track of its entire history of state roots and the hash of each batch. If anyone discovers that one batch had an incorrect post-state root, they can publish a proof to chain, proving that the batch was computed incorrectly. The contract verifies the proof, and reverts that batch and all batches after it.
2. **ZK rollups**, which use **validity proofs**: every batch includes a cryptographic proof called a ZK-SNARK (eg. using the PLONK protocol), which proves that the post-state root is the correct result of executing the batch. No matter how large the computation, the proof can be very quickly verified on-chain.

There are complex tradeoffs between the two flavors of rollups:

Property	Optimistic rollups	ZK rollups
Fixed gas cost per batch	**~40,000** (a lightweight transaction that mainly just changes the value of the state root)	~500,000 (verification of a ZK-SNARK is quite computationally intensive)
Withdrawal period	~1 week (withdrawals need to be delayed to give time for someone to publish a fraud proof and cancel the withdrawal if it is fraudulent)	**Very fast** (just wait for the next batch)
Complexity of technology	**Low**	High (ZK-SNARKs are very new and mathematically complex technology)
Generalizability	**Easier** (general-purpose EVM rollups are already close to mainnet)	Harder (ZK-SNARK proving general-purpose EVM execution is much harder than proving simple computations, though there are efforts (eg. Cairo) working to improve on this)
		Lower (if data in a transaction is only used to verify, and not to cause state changes, then this data

|		
Per-transaction on-chain gas costs	Higher	can be left out, whereas in an optimistic rollup it would need to be published in case it needs to be checked in a fraud proof)
Off-chain computation costs	**Lower** (though there is more need for many full nodes to redo the computation)	Higher (ZK-SNARK proving especially for general-purpose computation can be expensive, potentially many thousands of times more expensive than running the computation directly)

In general, my own view is that in the short term, optimistic rollups are likely to win out for general-purpose EVM computation and ZK rollups are likely to win out for simple payments, exchange and other application-specific use cases, but in the medium to long term ZK rollups will win out in all use cases as ZK-SNARK technology improves.

Anatomy of a fraud proof

The security of an optimistic rollup depends on the idea that if someone publishes an invalid batch into the rollup, *anyone else* who was keeping up with the chain and detected the fraud can publish a fraud proof, proving to the contract that that batch is invalid and should be reverted.
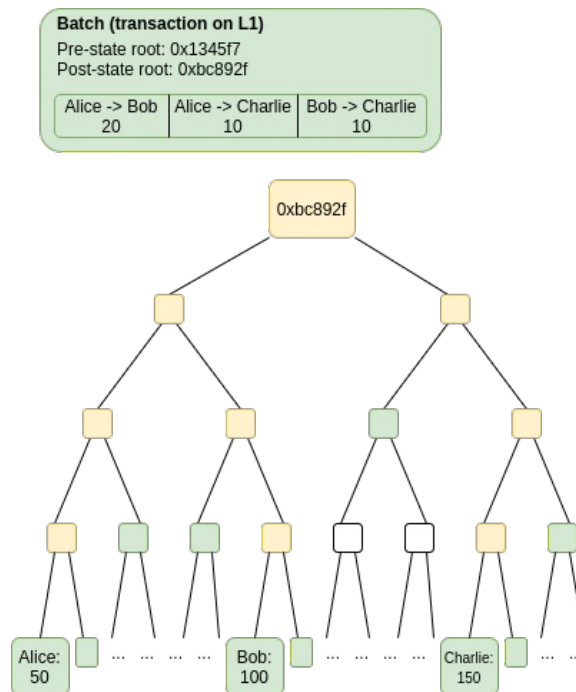
A fraud proof claiming that a batch was invalid would contain the data in green: the batch itself (which could be checked against a hash stored on chain) and the parts of the Merkle tree needed to prove just the specific accounts that were read and/or modified by the batch. The nodes in the tree in yellow can be reconstructed from the nodes in green and so do not need to be provided. This data is sufficient to execute the batch and compute the post-state root (note that this is exactly the same as how stateless clients verify individual blocks). If the computed post-state root and the provided post-state root in the batch are not the same, then the batch is fraudulent.

It is guaranteed that if a batch was constructed incorrectly, *and all previous batches were constructed correctly*, then it is possible to create a fraud proof showing the the batch was constructed incorrectly. Note the claim about previous batches: if there was more than one invalid batch published to the rollup, then it is best to try to prove the earliest one invalid. It is also, of course, guaranteed that if a batch was constructed correctly, then it is never possible to create a fraud proof showing that the batch is invalid.

How does compression work?

A simple Ethereum transaction (to send ETH) takes ~110 bytes. An ETH transfer on a rollup, however, takes only ~12 bytes:

Parameter	Ethereum	Rollup

Nonce	~3	0
Gasprice	~8	0-0.5
Gas	3	0-0.5
To	21	4
Value	~9	~3
Signature	~68 (2 + 33 + 33)	~0.5
From	0 (recovered from sig)	4
Total	~112	~12

Part of this is simply superior encoding: Ethereum's RLP wastes 1 byte per value on the length of each value. But there are also some very clever compression tricks that are going on:

- **Nonce**: the purpose of this parameter is to prevent replays. If the current nonce of an account is 5, the next transaction from that account must have nonce 5, but once the transaction is processed the nonce in the account will be incremented to 6 so the transaction cannot be processed again. In the rollup, we can omit the nonce entirely, because we just recover the nonce from the pre-state; if someone tries replaying a transaction with an earlier nonce, the signature would fail to verify, as the signature would be checked against data that contains the new higher nonce.
- **Gasprice**: we can allow users to pay with a fixed range of gasprices, eg. a choice of 16 consecutive powers of two. Alternatively, we could just have a fixed fee level in each batch, or even move gas payment outside the rollup protocol entirely and have transactors pay batch creators for inclusion through a channel.
- **Gas**: we could similarly restrict the total gas to a choice of consecutive powers of two. Alternatively, we could just have a gas limit only at the batch level.
- **To**: we can replace the 20-byte address with an *index* (eg. if an address is the 4527th address added to the tree, we just use the index 4527 to refer to it. We would add a subtree to the state to store the mapping of indices to addresses).
- **Value**: we can store value in scientific notation. In most cases, transfers only need 1-3 significant digits.
- **Signature**: we can use [BLS aggregate signatures](), which allows many signatures to be aggregated into a single ~32-96 byte (depending on protocol) signature. This signature can then be checked against the entire set of messages and senders in a batch all at once. The ~0.5 in the table represents the fact that there is a limit on how many signatures can be combined in an aggregate that can be verified in a single block, and so large batches would need one signature per ~100 transactions.

One important compression trick that is specific to ZK rollups is that if a part of a transaction is only used for verification, and is not relevant to computing the state update, then that part can be left off-chain. This cannot be done in an optimistic rollup because that data would still need to be included on-chain in case it needs to be later checked in a fraud proof, whereas in a ZK rollup the SNARK proving correctness of the batch already proves that any data needed for verification was provided. An important example of this is privacy-preserving rollups: in an optimistic rollup the ~500 byte ZK-SNARK used for privacy in each transaction needs to go on chain, whereas in a ZK rollup the ZK-SNARK covering the entire batch already leaves no doubt that the "inner" ZK-SNARKs are valid.

These compression tricks are key to the scalability of rollups; without them, rollups would be perhaps only a ~10x improvement on the scalability of the base chain (though there are some specific computation-heavy applications where even simple rollups are powerful), whereas with compression tricks the scaling factor can go over 100x for almost all applications.

Who can submit a batch?

There are a number of schools of thought for who can submit a batch in an optimistic or ZK rollup. Generally, everyone agrees that in order to be able to submit a batch, a user must put down a large deposit; if that user ever submits a fraudulent batch (eg. with an invalid state root), that deposit would be part burned and part given as a reward to the fraud prover. But beyond that, there are many possibilities:

- **Total anarchy**: anyone can submit a batch at any time. This is the simplest approach, but it has some important drawbacks. Particularly, there is a risk that multiple participants will generate and attempt to submit batches in parallel, and only one of those batches can be successfully included. This leads to a large amount of wasted effort in generating proofs and/or wasted gas in publishing batches to chain.
- **Centralized sequencer**: there is a single actor, the **sequencer**, who can submit batches (with an exception for withdrawals: the usual technique is that a user can first submit a withdrawal request, and then if the sequencer does not process that withdrawal in the next batch, then the user can submit a single-operation batch themselves). This is the most "efficient", but it is reliant on a central actor for liveness.
- **Sequencer auction**: an auction is held (eg. every day) to determine who has the right to be the sequencer for the next day. This technique has the advantage that it raises funds which could be distributed by eg. a DAO controlled by the rollup (see: [MEV auctions]())
- **Random selection from PoS set**: anyone can deposit ETH (or perhaps the rollup's own protocol token) into the rollup contract, and the sequencer of each batch is randomly selected from one of the depositors, with the probability of being selected being proportional to the amount deposited. The main

drawback of this technique is that it leads to large amounts of needless capital lockup.

- **DPoS voting**: there is a single sequencer selected with an auction but if they perform poorly token holders can vote to kick them out and hold a new auction to replace them.

Split batching and state root provision

Some of the rollups being currently developed are using a "split batch" paradigm, where the action of submitting a batch of layer-2 transactions and the action of submitting a state root are done separately. This has some key advantages:

1. You can allow many sequencers in parallel to publish batches in order to improve censorship resistance, without worrying that some batches will be invalid because some other batch got included first.
2. If a state root is fraudulent, you don't need to revert the entire batch; you can revert just the state root, and wait for someone to provide a new state root for the same batch. This gives transaction senders a better guarantee that their transactions will not be reverted.

So all in all, there is a fairly complex zoo of techniques that are trying to balance between complicated tradeoffs involving efficiency, simplicity, censorship resistance and other goals. It's still too early to say which combination of these ideas works best; time will tell.

How much scaling do rollups give you?

On the existing Ethereum chain, the gas limit is 12.5 million, and each byte of data in a transaction costs 16 gas. This means that if a block contains nothing but a single batch (we'll say a ZK rollup is used, spending 500k gas on proof verification), that batch can have (12 million / 16) = 750,000 bytes of data. As shown above, a rollup for ETH transfers requires only 12 bytes per user operation, meaning that the batch can contain up to 62,500 transactions. At an average block time of [13 seconds](), this translates to ~4807 TPS (compared to 12.5 million / 21000 / 13 ~= 45 TPS for ETH transfers directly on Ethereum itself).

Here's a chart for some other example use cases:

Application	Bytes in rollup	Gas cost on layer 1	Max scalability gain
ETH transfer	**12**	21,000	105x
ERC20 transfer	**16** (4 more bytes to specify which token)	~50,000	187x
Uniswap trade	**~14** (4 bytes sender + 4 bytes recipient + 3 bytes value + 1 byte max price + 1 byte misc)	~100,000	428x
Privacy-preserving withdrawal (Optimistic rollup)	**296** (4 bytes index of root + 32 bytes nullifier + 4 bytes recipient + 256 bytes ZK-SNARK proof)	[~380,000]()	77x
Privacy-preserving withdrawal (ZK rollup)	**40** (4 bytes index of root + 32 bytes nullifier + 4 bytes recipient)	[~380,000]()	570x

*Max scalability gain is calculated as (L1 gas cost) / (bytes in rollup * 16) * 12 million / 12.5 million.*

Now, it is worth keeping in mind that these figures are overly optimistic for a few reasons. Most importantly, a block would almost never just contain one batch, at the very least because there are and will be multiple rollups. Second, deposits and withdrawals will continue to exist. Third, *in the short term* usage will be low, and so fixed costs will dominate. But even with these factors taken into account, scalability gains of over 100x are expected to be the norm.

Now what if we want to go above ~1000-4000 TPS (depending on the specific use case)? Here is where [eth2 data sharding]() comes in. The sharding proposal opens up a space of 16 MB every 12 seconds that can be filled with any data, and the system guarantees consensus on the availability of that data. This data space can be used by rollups. This ~1398k bytes per sec is a 23x improvement on the ~60 kB/sec of the existing Ethereum chain, and in the longer term the data capacity is expected to grow even further. Hence, rollups that use eth2 sharded data can collectively process as much as ~100k TPS, and even more in the future.

What are some not-yet-fully-solved challenges in rollups?

While the basic concept of a rollup is now well-understood, we are quite certain that they are fundamentally feasible and secure, and multiple rollups have already been deployed to mainnet, there are still many areas of rollup design that have not been well explored, and quite a few challenges in fully bringing large parts of the Ethereum ecosystem onto rollups to take advantage of their scalability. Some key challenges include:

- **User and ecosystem onboarding** - not many applications use rollups, rollups are unfamiliar to users,

and few wallets have started integrating rollups. Merchants and charities do not yet accept them for payments.

- **Cross-rollup transactions** - efficiently moving assets and data (eg. oracle outputs) from one rollup into another without incurring the expense of going through the base layer.
- **Auditing incentives** - how to maximize the chance that at least one honest node actually will be fully verifying an optimistic rollup so they can publish a fraud proof if something goes wrong? For small-scale rollups (up to a few hundred TPS) this is not a significant issue and one can simply rely on altruism, but for larger-scale rollups more explicit reasoning about this is needed.
- **Exploring the design space in between plasma and rollups** - are there techniques that put *some* state-update-relevant data on chain but not *all* of it, and is there anything useful that could come out of that?
- **Maximizing security of pre-confirmations** - many rollups provide a notion of "pre-confirmation" for faster UX, where the sequencer immediately provides a promise that a transaction will be included in the next batch, and the sequencer's deposit is destroyed if they break their word. But the economy security of this scheme is limited, because of the possibility of making many promises to very many actors at the same time. Can this mechanism be improved?
- **Improving speed of response to absent sequencers** - if the sequencer of a rollup suddenly goes offline, it would be valuable to recover from that situation maximally quickly and cheaply, either quickly and cheaply mass-exiting to a different rollup or replacing the sequencer.
- **Efficient ZK-VM** - generating a ZK-SNARK proof that general-purpose EVM code (or some different VM that existing smart contracts can be compiled to) has been executed correctly and has a given result.

Conclusions

Rollups are a powerful new layer-2 scaling paradigm, and are expected to be a cornerstone of Ethereum scaling in the short and medium-term future (and possibly long-term as well). They have seen a large amount of excitement from the Ethereum community because unlike previous attempts at layer-2 scaling, they can support general-purpose EVM code, allowing existing applications to easily migrate over. They do this by making a key compromise: not trying to go fully off-chain, but instead leaving a small amount of data per transaction on-chain.

There are many kinds of rollups, and many choices in the design space: one can have an optimistic rollup using fraud proofs, or a ZK rollup using validity proofs (aka. ZK-SNARKs). The sequencer (the user that can publish transaction batches to chain) can be either a centralized actor, or a free-for-all, or many other choices in between. Rollups are still an early-stage technology, and development is continuing rapidly, but they work and some (notably [Loopring](#), [ZKSync](#) and [DeversiFi](#)) have already been running for months. Expect much more exciting work to come out of the rollup space in the years to come.

Endnotes on 2020: Crypto and Beyond

2020 Dec 28

I'm writing this sitting in Singapore, the city in which I've now spent nearly half a year of uninterrupted time - an unremarkable duration for many, but for myself the longest I've stayed in any one place for nearly a decade. After months of fighting what may perhaps even be humanity's first boss-level enemy since 1945, the city itself is close to normal, though the world as a whole and its 7.8 billion inhabitants, normally so close by, continue to be so far away. Many other parts of the world have done less well and suffered more, though there is now a light at the end of the tunnel, as hopefully rapid deployment of vaccines will help humanity as a whole overcome this great challenge.

2020 has been a strange year because of these events, but also others. As life "away from keyboard (AFK)" has gotten much more constrained and challenging, the internet has been supercharged, with consequences both good and bad. Politics around the world has gone in strange directions, and I am continually worried by the many political factions that are so easily abandoning their most basic principles because they seem to have decided that their (often mutually contradictory) personal causes are just too important. And yet at the same time, there are rays of hope coming from unusual corners, with new technological discoveries in transportation, medicine, artificial intelligence - and, of course, blockchains and cryptography - that could open up a new chapter for humanity finally coming to fruition.

Where we started   *Where we're going*

And so, 2020 is as good a year as any to ponder a key question: how should we re-evaluate our models of the world? What ways of seeing, understanding and reasoning about the world are going to be more useful in the decades to come, and what paths are no longer as valuable? What paths did we not see before that were valuable all along? In this post, I will give some of my own answers, covering far from everything but digging into a few specifics that seem particularly interesting. It's sometimes hard to tell which of these ideas are a recognition of a changing reality, and which are just myself finally seeing what has always been there; often enough it's some of both. The answers to these questions have a deep relevance both to the crypto space that I call home as well as to the wider world.

The Changing Role of Economics

Economics has historically focused on "goods" in the form of physical objects: production of food, manufacturing of widgets, buying and selling houses, and the like. Physical objects have some particular properties: they can be transferred, destroyed, bought and sold, but not copied. If one person is using a physical object, it's usually impractical for another person to use it simultaneously. Many objects are only valuable if "consumed" outright. Making ten copies of an object requires something close to ten times the resources that it takes to make one (not quite ten times, but surprisingly close, especially at larger scales). **But on the internet, very different rules apply.**

Copying is cheap. I can write an article or a piece of code once, and it usually takes quite a bit of effort to write it once, but once that work is done, an unlimited number of people can download and enjoy it. Very few things are "consumable"; often products are superseded by better ones, but if that does not happen, something produced today may continue to provide value to people until the end of time.

On the internet, "public goods" take center stage. Certainly, private goods exist, particularly in the form of individuals' scarce attention and time and virtual assets that command that attention, but the *average* interaction is one-to-many, not one-to-one. Confounding the situation even further, the "many" rarely maps easily to our traditional structures for structuring one-to-many interactions, such as companies, cities or countries;. Instead, these public goods are typically public across a widely scattered collection of people all around the world. Many online platforms serving wide groups of people need governance, to decide on features, content moderation policies or other challenges important to their user community, though there too, the user community rarely maps cleanly to anything but itself. How is it fair for the US government to govern Twitter, when Twitter is often a platform for public debates between US politicians and representatives of its geopolitical rivals? But clearly, governance challenges exist - and so we need more creative solutions.

This is not merely of interest to "pure" online services. Though goods in the physical world - food, houses, healthcare, transportation - continue to be as important as ever, *improvements* in these goods depend even more than before on technology, and technological progress *does* happen over the internet.

CLR MATCHING ROUND 7 — 💡 Infra Grants — GITCOIN	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MACTHING
1 DAppNode - Panvala League	217	$5,514	$22,089
2 White Hat Hacking	199	$17,462	$21,796
3 Turbo-Geth	181	$12,383	$14,794
4 WalletConnect	169	$6,462	$13,313
5 ArchiveNode.io - The Public Access Ethereum Archive Node	160	$9,486	$10,767
6 Fuel Labs	147	$8,273	$8,838
7 Ethereum on ARM	136	$2,676	$6,350
8 Nethermind	116	$4,249	$5,054
9 Create Eth App	98	$1,275	$4,138
10 Ethereum Swarm	100	$888	$3,763

CLR MATCHING ROUND 7 — ❤️ Community Grants — GITCOIN	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MACTHING
1 Bankless	292	$6,869	$40,751
2 Week in Ethereum News	213	$3,184	$20,152
3 DeFi Dad DeFi Tutorials	210	$2,699	$17,691
4 EthHub - Ethereum Information Hub	191	$2,964	$14,259
5 Ethereum Magicians	165	$1,746	$12,020
6 Zero Knowledge Podcast	150	$2,059	$9,376
7 The Defiant	135	$3,483	$6,559
8 Meta Gamma Delta - Panvala League	82	$1,442	$3,580
9 RadicalxChange Foundation	87	$1,359	$3,572
10 ETH Memes Twitter	93	$1,005	$2,913

Examples of important public goods in the Ethereum ecosystem that were funded by the [recent Gitcoin quadratic funding round](). Open source software ecosystems, including blockchains, are hugely dependent on public goods.

But also, **economics itself seems to be a less powerful tool in dealing with these issues**. Out of all the challenges of 2020, how many can be understood by looking at supply and demand curves? One way to see what is going on here is by looking at the relationship between *economics* and *politics*. In the 19th century, the two were frequently viewed as being tied together, a subject called "political economy". In the 20th century, the two are more typically split apart. But in the 21st century, the lines between "private" and "public" are once again rapidly blurring. Governments are behaving more like market actors, and corporations are behaving more like governments.

We see this merge happening in the crypto space as well, as the researchers' eye of attention is increasingly switching focus to the challenge of governance. Five years ago, the main economic topics being considered in the crypto space had to do with consensus theory. This is a tractable economics problem with clear goals, and so we would on several occasions obtain nice clean results like the selfish mining paper. Some points of subjectivity, like quantifying decentralization, exist, but they could be easily encapsulated and treated separately from the formal math of the mechanism design. But in the last few years, we have seen the rise of increasingly complicated financial protocols and DAOs on top of blockchains, and at the same time governance challenges *within* blockchains. Should Bitcoin Cash redirect 12.5% of its block reward toward paying a developer team? If so, who decides who that developer team is? Should Zcash extend its 20% developer reward for another four years? These problems certainly can be analyzed economically to some extent, but the analysis inevitably gets stuck at concepts like coordination, flipping between equilibria, "Schelling points" and "legitimacy", that are much more difficult to express with numbers. And so, a hybrid discipline, combining formal mathematical reasoning with the softer style of humanistic reasoning, is required.

We wanted digital nations, instead we got digital nationalism

One of the most fascinating things that I noticed fairly early in the crypto space starting from around 2014 is just how quickly it started replicating the political patterns of the world at large. I don't mean this just in some broad abstract sense of "people are forming tribes and attacking each other", I mean similarities that are surprisingly deep and specific.

First, the story. From 2009 to about 2013, the Bitcoin world was a relatively innocent happy place. The community was rapidly growing, prices were rising, and disagreements over block size or long-term direction, while present, were largely academic and took up little attention compared to the shared broader goal of helping Bitcoin grow and prosper.

But in 2014, the schisms started to arise. Transaction volumes on the Bitcoin blockchain hit 250 kilobytes per block and kept rising, for the first time raising fears that blockchain usage might actually hit the 1 MB limit before the limit could be increased. Non-Bitcoin blockchains, up until this point a minor sideshow, suddenly became a major part of the space, with Ethereum itself arguably leading the charge. And it was during these events that disagreements that were before politely hidden beneath the surface suddenly blew up. "Bitcoin maximalism", the idea that the goal of the crypto space should not be a diverse ecosystem of cryptocurrencies generally but Bitcoin and Bitcoin alone specifically, grew from a niche curiosity into a prominent and angry movement that Dominic Williams and I quickly saw for what it is and gave its current name. The small block ideology, arguing that the block size should be increased very slowly or even never increased at all regardless of how high transaction fees go, began to take root.

The disagreements within Bitcoin would soon turn into an all-out civil war. Theymos, the operator of the /r/bitcoin subreddit and several other key public Bitcoin discussion spaces, resorted to extreme censorship to impose his (small-block-leaning) views on the community. In response, the big-blockers moved to a new subreddit, /r/btc. Some valiantly attempted to defuse tensions with diplomatic conferences including a famous one in Hong Kong, and a seeming consensus was reached, though one year later the small block side would end up reneging on its part of the deal. By 2017, the big block faction was firmly on its way to defeat, and in August of that year they would secede (or "fork off") to implement their own vision on their own separate continuation of the Bitcoin blockchain, which they called "Bitcoin Cash" (symbol BCH).

The community split was chaotic, and one can see this in how the channels of communication were split up in the divorce: /r/bitcoin stayed under the control of supporters of Bitcoin (BTC). /r/btc was controlled by supporters of Bitcoin Cash (BCH). Bitcoin.org was controlled by supporters of Bitcoin (BTC). Bitcoin.com on the other hand was controlled by supporters of Bitcoin Cash (BCH). Each side claimed themselves to be the true Bitcoin. The result looked remarkably similar to one of those civil wars that happens from time to time that results in a country splitting in half, the two halves calling themselves almost identical names that differ only in which subset of the words "democratic", "people's" and "republic" appears on each side. Neither side had the ability to destroy the other, and of course there was no higher authority to adjudicate the dispute.
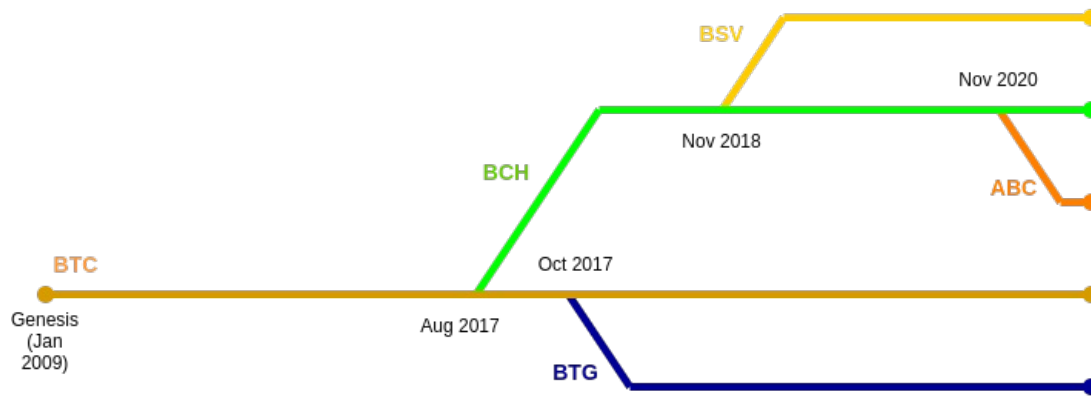
Major Bitcoin forks, as of 2020. Does not include Bitcoin Diamond, Bitcoin Rhodium, Bitcoin Private, or any of the other long list of Bitcoin forks that I would highly recommend you just ignore completely, except to sell (and perhaps you should sell some of the forks listed above too, eg. [BSV is definitely a scam!](#))

Around the same time, Ethereum had its own chaotic split, in the form of [the DAO fork](#), a highly controversial resolution to a theft in which over $50 million was stolen from the first major smart contract application on Ethereum. Just like in the Bitcoin case, there was first a civil war - though only lasting four weeks - and then a chain split, followed by an online war between the two now-separate chains, Ethereum (ETH) and Ethereum Classic (ETC). The naming split was as fun as in Bitcoin: the Ethereum Foundation held [ethereumproject on Twitter](#) but Ethereum Classic supporters held [ethereumproject on Github](#).

Some on the Ethereum side would argue that Ethereum Classic had very few "real" supporters, and the whole thing was mostly a social attack by Bitcoin supporters: either to support the version of Ethereum that aligned with their values, or to cause chaos and destroy Ethereum outright. I myself believed these claims somewhat at the beginning, though over time I came to realize that they were overhyped. It is true that some Bitcoin supporters had certainly tried to shape the outcome in their own image. But to a large extent, as is the case in many conflicts, the "foreign interference" card was simply a psychological defense that many Ethereum supporters, myself included, subconsciously used to shield ourselves from the fact that many people within our own community really did have different values. Fortunately relations between the two currencies have since improved - in part thanks to the excellent diplomatic skills of [Virgil Griffith](#) - and Ethereum Classic developers have even agreed to move to a different Github page.

Civil wars, [alliances](#), blocs, alliances with participants in civil wars, you can all find it in crypto. Though fortunately, the conflict is all virtual and online, without the extremely harmful in-person consequences that often come with such things happening in real life. So what can we learn from all this? One important takeaway is this: if phenomena like this happen in contexts as widely different from each other as conflicts between countries, conflicts between religions and relations within and between purely digital cryptocurrencies, then perhaps what we're looking at is the indelible epiphenomena of human nature - something much more difficult to resolve than by changing what kinds of groups we organize in. So we should expect situations like this to continue to play out in many contexts over the decades to come. And perhaps it's harder than we thought to separate the good that may come out of this from the bad: those same energies that drive us to fight also drive us to contribute.

What motivates us anyway?

One of the key intellectual undercurrents of the 2000s era was the recognition of the importance of non-monetary motivations. People are motivated not just by earning as much money as possible in the work and extracting enjoyment from their money in their family lives; even at work we are motivated by social status, honor, altruism, reciprocity, a feeling of contribution, different social conceptions of what is good and valuable, and much more.

These differences are very meaningful and measurable. For one example, see [this Swiss study](#) on compensating differentials for immoral work - how much extra do employers have to pay to convince someone to do a job if that job is considered morally unsavory?

Figure 1: Correlation between wages and perceived industry immorality
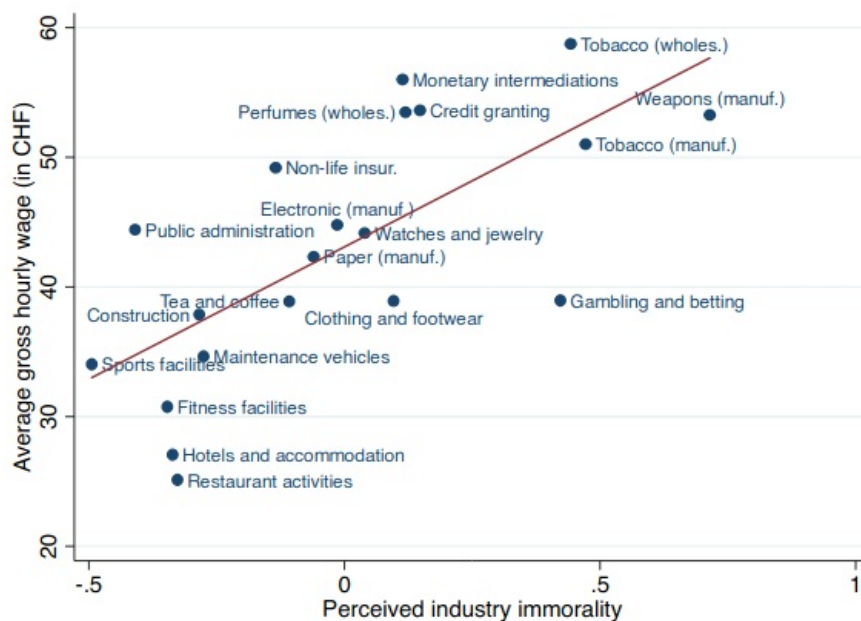
Source: Weighted data from the SLFS, years 2010-2016 (wage) and our own survey (perceived industry immorality). Notes: Perceived immorality is in [-1, 1] where -1 means very moral, 0 means neutral and 1 means very immoral. Real gross hourly wage in 2010 CHF. N = 32,638.

As we can see, the effects are massive: if a job is widely considered immoral, you need to pay employees almost twice as much for them to be willing to do it. From personal experience, I would even argue that this understates the case: in many cases, top-quality workers would not be willing to work for a company that they think is bad for the world at almost any price. "Work" that is difficult to formalize (eg. word-of-mouth marketing) functions similarly: if people think a project is good, they will do it for free, if they do not, they will not do it at all. This is also likely why blockchain projects that raise a lot of money but are unscrupulous, or even just corporate-controlled profit-oriented "VC chains", tend to fail: even a billion dollars of capital cannot compete with a project *having a soul*.

That said, it is possible to be overly idealistic about this fact, in several ways. First of all, **while this decentralized, non-market, non-governmental subsidy toward projects that are socially considered to be good is massive, likely amounting to tens of trillions of dollars per year globally, its effect is not infinite**. If a developer has a choice between earning $30,000 per year by being "ideologically pure", and making a $30 million ICO by sticking a needless token into their project, they *will* do the latter. Second, idealistic motivations are *uneven* in what they motivate. Rick Falkvinge's Swarmwise played up the possibility of decentralized non-market organization in part by pointing to political activism as a key example. And this is true, political activism does not require getting paid. But longer and more grueling tasks, even something as simple as making good user interfaces, are not so easily intrinsically motivated. And so if you rely on intrinsic motivation too much, you get projects where some tasks are overdone and other tasks are done poorly, or even ignored entirely. And third, perceptions of what people find intrinsically attractive to work on may change, and may even be manipulated.

One important conclusion for me from this is the importance of culture (and that oh-so-important word that crypto influencers have unfortunately ruined for me, "*narrative*"). If a project having a high moral standing is equivalent to that project having twice as much money, or even more, then culture and narrative are extremely powerful forces that command the equivalent of tens of trillions of dollars of value. And this does not even begin to cover the role of such concepts in shaping our perceptions of legitimacy and coordination. And so anything that influences the culture can have a great impact on the world and on people's financial interests, and we're going to see more and more sophisticated efforts from all kinds of actors to do so systematically and deliberately. This is the darker conclusion of the importance of non-monetary social motivations - they create the battlefield for the permanent and final frontier of war, the war that is fortunately not usually deadly but unfortunately impossible to create peace treaties for because of how inextricably subjective it is to determine what even counts as a battle: the culture war.

Big X is here to stay, for all X

One of the great debates of the 20th century is that between "Big Government" and "Big Business" - with various permutations of each: Big Brother, Big Banks, Big Tech, also at times joining the stage. In this environment, the Great Ideologies were typically defined by trying to abolish the Big X that they *disliked*: communism focusing on corporations, anarcho-capitalism on governments, and so forth. Looking back in 2020, one may ask: which of the Great Ideologies succeeded, and which failed?

Let us zoom into one specific example: the 1996 [Declaration of Independence of Cyberspace](#):

> Governments of the Industrial World, you weary giants of flesh and steel, I come from Cyberspace, the new home of Mind. On behalf of the future, I ask you of the past to leave us alone. You are not welcome among us. You have no sovereignty where we gather.

And the similarly-spirited [Crypto-Anarchist Manifesto](#):

> Computer technology is on the verge of providing the ability for individuals and groups to communicate and interact with each other in a totally anonymous manner. Two persons may exchange messages, conduct business, and negotiate electronic contracts without ever knowing the True Name, or legal identity, of the other. Interactions over networks will be untraceable, via extensive re-routing of encrypted packets and tamper-proof boxes which implement cryptographic protocols with nearly perfect assurance against any tampering. Reputations will be of central importance, far more important in dealings than even the credit ratings of today. These developments will alter completely the nature of government regulation, the ability to tax and control economic interactions, the ability to keep information secret, and will even alter the nature of trust and reputation.

How have these predictions fared? The answer is interesting: I would say that they succeeded in one part and failed in the other. What succeeded? We have interactions over networks, we have powerful cryptography that is difficult for even state actors to break, we even have powerful *cryptocurrency*, with smart contract capabilities that the thinkers of the 1990s mostly did not even anticipate, and we're increasingly moving toward anonymized reputation systems [with zero knowledge proofs](#). What failed? Well, the government did not go away. And what just proved to be totally unexpected? Perhaps the most interesting plot twist is that the two forces are, a few exceptions notwithstanding, by and large *not* acting like mortal enemies, and there are even many people within governments that are earnestly trying to find ways to be friendly to blockchains and cryptocurrency and new forms of cryptographic trust.

What we see in 2020 is this: Big Government is as powerful as ever, but Big Business is *also* as powerful as ever. "[Big Protest Mob](#)" is as powerful as ever too, as is Big Tech, and soon enough perhaps Big Cryptography. It's a densely populated jungle, with an uneasy peace between many complicated actors. If you define success as the total absence of a category of powerful actor or even a category of activity that you dislike, then you will probably leave the 21st century disappointed. But if you define success more through what happens than through what doesn't happen, and you are okay with imperfect outcomes, there is enough space to make everyone happy.

Often, the boundary between multiple intersecting worlds is the most interesting place to be. The monkeys get it.

Prospering in the dense jungle

So we have a world where:

- One-to-one interactions are less important, one-to-many and many-to-many interactions are more important.
- The environment is much more *chaotic*, and difficult to model with clean and simple equations. Many-to-many interactions particularly follow strange rules that we still do not understand well.
- The environment is *dense,* and different categories of powerful actors are forced to live quite closely side by side with each other.

In some ways, this is a world that is less convenient for someone like myself. I grew up with a form of economics that is focused on analyzing simpler physical objects and buying and selling, and am now forced to contend with a world where such analysis, while not irrelevant, is significantly less relevant than before. That said, transitions are always challenging. In fact, transitions are particularly challenging for those who think that they are not challenging because they think that the transition merely confirms what they believed all along. If you are still operating today precisely according to a script that was created in 2009, when the Great Financial Crisis was the most recent pivotal event on anyone's mind, then there are almost certainly important things that happened in the last decade that you are missing. An ideology that's finished is an ideology that's dead.

It's a world where blockchains and cryptocurrencies are well poised to play an important part, though for reasons much more complex than many people think, and having as much to do with cultural forces as anything financial (one of the more underrated bull cases for cryptocurrency that I have always believed is simply the fact that *gold is lame*, the younger generations realize that it's lame, and that [$9 trillion](#) has to go *somewhere*). Similarly complex forces are what will lead to blockchains and cryptocurrencies being *useful*. It's easy to *say* that any application can be done more efficiently with a centralized service, but in practice social coordination problems are very real, and unwillingness to sign onto a system that has even a *perception* of non-neutrality or ongoing dependence on a third party is real too. And so the centralized and even consortium-based approaches claiming to replace blockchains don't get anywhere, while "dumb and inefficient" public-blockchain-based solutions just keep quietly moving forward and gaining actual adoption.

And finally it's a very multidisciplinary world, one that is much harder to break up into layers and analyze each layer separately. You may need to switch from one style of analysis to another style of analysis in mid-sentence. Things happen for strange and inscrutable reasons, and there are always surprises. The question that remains is: how do we adapt to it?

Convex and Concave Dispositions

2020 Nov 08

One of the major philosophical differences that I have noticed in how people approach making large-scale decisions in the world is how they approach the age-old tradeoff of compromise versus purity. Given a choice between two alternatives, often both expressed as deep principled philosophies, do you naturally gravitate toward the idea that one of the two paths should be correct and we should stick to it, or do you prefer to find a way in the middle between the two extremes?

In mathematical terms, we can rephrase this as follows: do you expect the world that we are living in, and in particular the way that it responds to the actions that we take, to fundamentally be **concave** or **convex**?
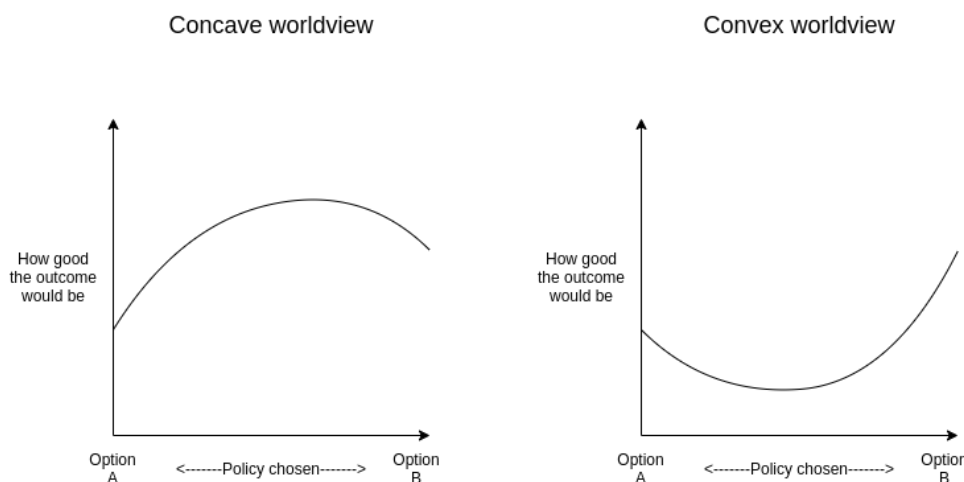
Someone with a concave disposition might say things like this:

- "Going to the extremes has never been good for us; you can die from being too hot or too cold. We need to find the balance between the two that's just right"
- "If you implement only a little bit of a philosophy, you can pick the parts that have the highest benefits and the lowest risks, and avoid the parts that are more risky. But if you insist on going to the extremes, once you've picked the low-hanging fruit, you'll be forced to look harder and harder for smaller and smaller benefits, and before you know it the growing risks might outweigh the benefit of the whole thing"
- "The opposing philosophy probably has some value too, so we should try to combine the best parts of both, and definitely avoid doing things that the opposing philosophy considers to be *extremely* terrible, just in case"

Someone with a convex disposition might say things like this:

- "We need to focus. Otherwise, we risk becoming a jack of all trades, master of none"
- "If we take even a few steps down that road, it will become slippery slope and only pull us down ever further until we end up in the abyss. There's only two stable positions on the slope: either we're down there, or we stay up here"
- "If you give an inch, they will take a mile"
- "Whether we're following this philosophy or that philosophy, we should be following *some* philosophy and just stick to it. Making a wishy-washy mix of everything doesn't make sense"

I personally find myself perenially more sympathetic to the concave approach than the convex approach, across a wide variety of contexts. If I had to choose either (i) a coin-flip between anarcho-capitalism and Soviet communism or (ii) a 50/50 compromise between the two, I would pick the latter in a heartbeat. I argued for moderation in Bitcoin block size debates, arguing against both 1-2 MB small blocks and 128 MB "very big blocks". I've argued against the idea that freedom and decentralization are "you either have it or you don't" properties with no middle ground. I argued in favor of the DAO fork, but to many people's surprise I've argued since then against similar "state-intervention" hard forks that were proposed more recently. As I said in 2019, "support for Szabo's law [blockchain immutability] is a spectrum, not a binary".

But as you can probably tell by the fact that I needed to make those statements at all, not everyone seems to share the same broad intuition. **I would particularly argue that the Ethereum ecosystem in general has a fundamentally concave temperament, while the Bitcoin ecosystem's temperament is much more fundamentally convex**. In Bitcoin land, you can frequently hear arguments that, for example, either you have self-sovereignty or you don't, or that any system must have either a fundamentally centralizing or a fundamentally decentralizing tendency, with no possibility halfway in between.

The occasional half-joking [support for Tron](#) is a key example: from my own concave point of view, if you value decentralization and immutability, you should recognize that while the Ethereum ecosystem does sometimes violate purist conceptions of these values, Tron violates them far more egregiously and without remorse, and so Ethereum is still by far the more palatable of the two options. But from a convex point of view, the extremeness of Tron's violations of these norms is a virtue: while Ethereum half-heartedly pretends to be decentralized, Tron is centralized *but at least it's proud and honest about it*.

This difference between concave and convex mindsets is not at all limited to arcane points about efficiency/decentralization tradeoffs in cryptocurrencies. It applies to politics (guess which side has more outright anarcho-capitalists), other choices in technology, and even what food you eat.
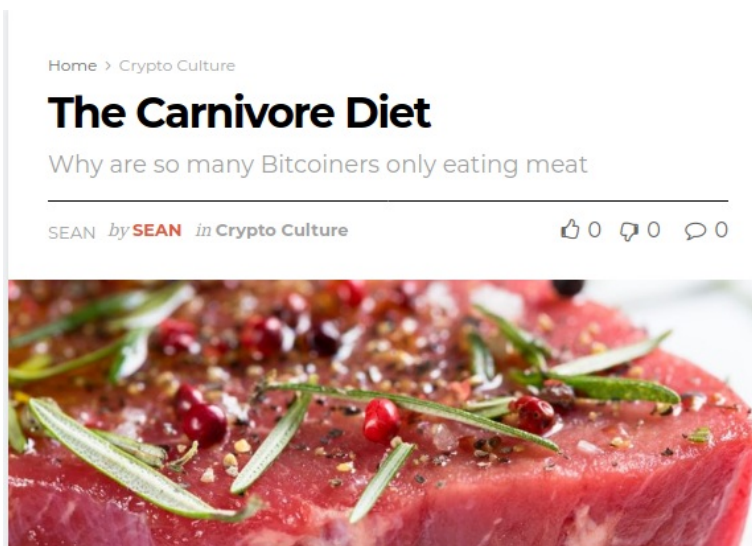
But in all of these questions too, I personally find myself fairly consistently coming out on the side of balance.

Being concave about concavity

But it's worth noting that [even on the meta-level](#), concave temperament is something that one must take great care to avoid being extreme about. There are certainly situations where policy A gives a good result, policy B gives a worse but still tolerable result, but a half-hearted mix between the two is worst of all. The coronavirus is perhaps an excellent example: a 100% effective travel ban is [far more than twice as useful](#) as a 50% effective travel ban. An effective lockdown that pushes the R0 of the virus down below 1 can eradicate the virus, leading to a quick recovery, but a half-hearted lockdown that only pushes the R0 down to 1.3 leads to months of agony with little to show for it. This is one possible explanation for why many Western countries responded poorly to it: political systems designed for compromise risk falling into middle approaches even when they are not effective.

Another example is a war: if you invade country A, you conquer country A, if you invade country B, you conquer country B, but if you invade both at the same time sending half your soldiers to each one, the power of the two combined will crush you. In general, problems where the effect of a response is convex are often places where you can find benefits of some degree of centralization.

But there are also many places where a mix is clearly better than either extreme. A common example is the question of setting tax rates. In economics there is the general principle that [deadweight loss is quadratic](#): that is, the harms from the inefficiency of a tax are proportional *to the square* of the tax rate. The reason why this is the case can be seen as follows. A tax rate of 2% deters very few transactions, and even the transactions it deters are not very valuable - how valuable can a transaction be if a mere 2% tax is enough to discourage the participants from making it? A tax rate of 20% would deter perhaps ten times more transactions, but *each individual transaction that was deterred is itself ten times more valuable to its participants* than in the 2% case. Hence, a 10x higher tax may cause 100x higher economic harm. And for this reason, a low tax is generally better than a coin flip between high tax and no tax.

By similar economic logic, an outright prohibition on some behavior may cause more than twice as much harm as a tax set high enough to only deter half of people from participating. Replacing existing prohibitions with medium-high punitive taxes (a very concave-temperamental thing to do) could increase efficiency, increase freedom *and* provide valuable revenue to build public goods or help the impoverished.

Another example of effects like this in [Laffer curve](#): a tax rate of zero raises no revenue, a tax rate of 100% raises no revenue because no one bothers to work, but some tax rate in the middle raises the most revenue. There are debates about what that revenue-maximizing rate is, but in general there's broad agreement that the chart looks something like this:
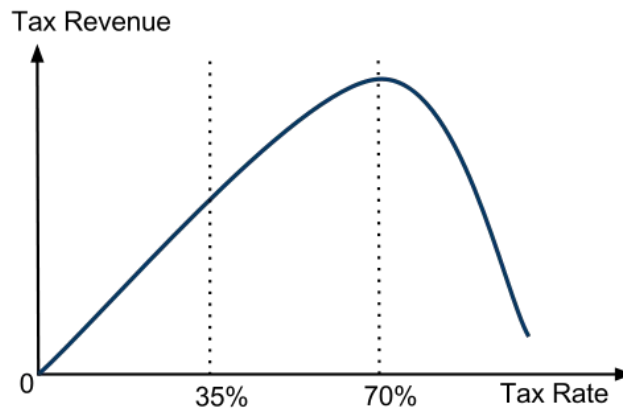
If you had to pick either the average of two proposed tax plans, or a coin-flip between them, it's obvious that the average is usually best. And taxes are not the only phenomenon that are like this; economics studies a wide array of "diminishing returns" phenomena which occur everywhere in production, consumption and many other aspects of regular day-to-day behavior. Finally, a common flip-side of diminishing returns is accelerating costs: to give one notable example, if you take standard economic models of utility of money, they directly imply that double the economic inequality can cause four times the harm.

The world has more than one dimension

Another point of complexity is that in the real world, policies are not just single-dimensional numbers. There are many ways to average between two different policies, or two different philosophies. One easy example to see this is: suppose that you and your friend want to live together, but you want to live in Toronto and your friend wants to live in New York. How would you compromise between these two options?

Well, you could take the geographic compromise, and enjoy your peaceful existence at the arithmetic midpoint between the two lovely cities at....
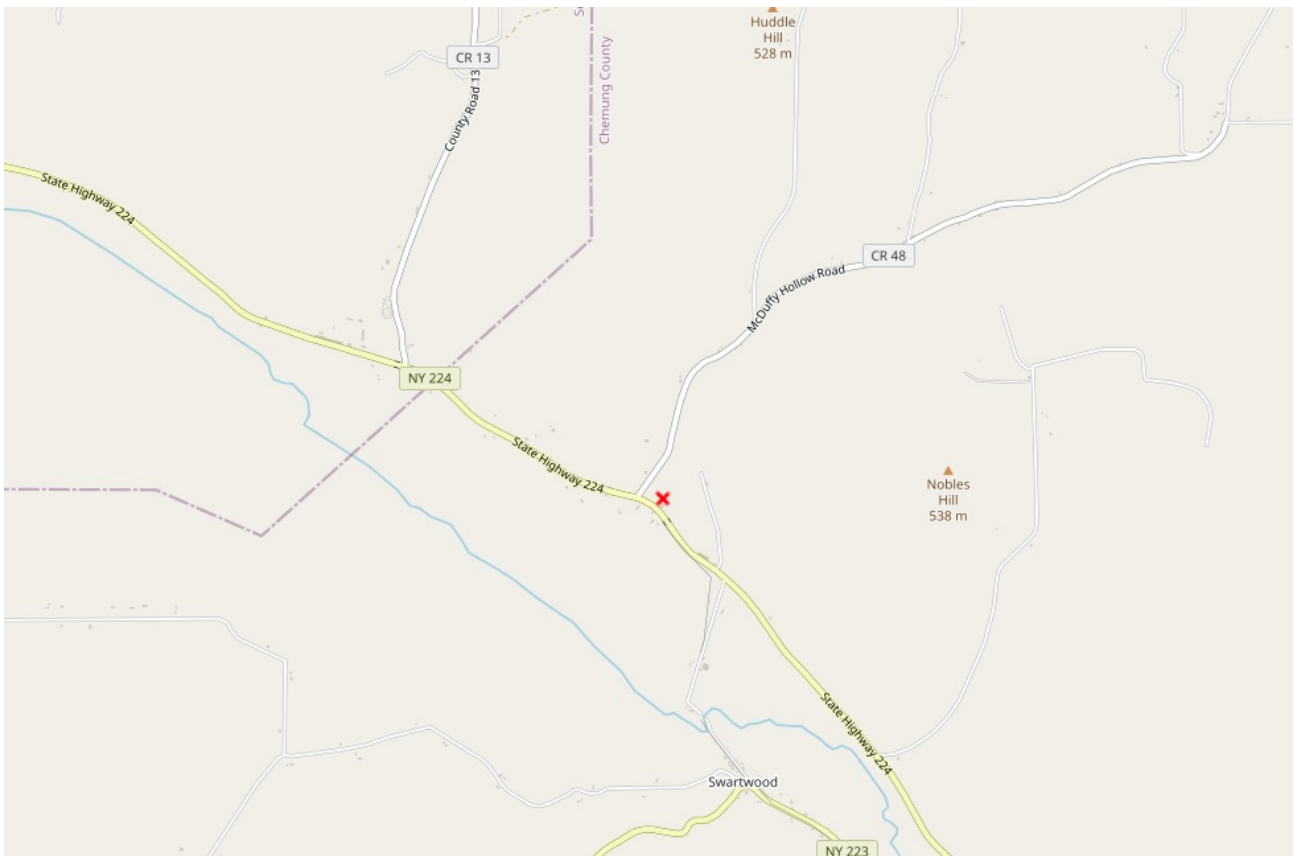
This Assembly of God church about 29km southwest of Ithaca, NY.

Or you could be even more mathematically pure, and take the straight-line midpoint between Toronto and New York without even bothering to stay on the Earth's surface. Then, you're still pretty close to that church, but you're six kilometers under it. A different way to compromise is spending six months every year in Toronto and six months in New York - and this may well be an actually reasonable path for some people to take.

The point is, when the options being presented to you are more complicated than simple single-dimensional numbers, figuring out *how* to compromise between the options well, and really take the best parts of both and not the worst parts of both, is an art, and a challenging one.

And this is to be expected: "convex" and "concave" are terms best suited to mathematical functions where the input and the output are both one-dimensional. The real world is high-dimensional - and as machine-learning researchers have [now well established](#), in high-dimensional environments the most common setting that you can expect to find yourself in is not a universally convex or universally concave one, but rather a *saddle point*: a point where the local region is convex in some directions but concave in other directions.
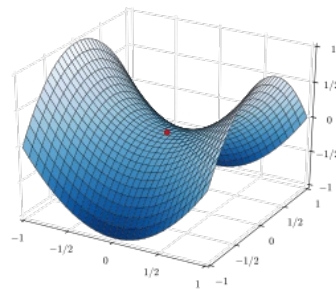
A saddle point. Convex left-to-right, concave forward-to-backward.

This is probably the best mathematical explanation for why both of these dispositions are to some extent necessary: the world is not entirely convex, but it is not entirely concave either. But the existence of *some* concave path between any two distant positions A and B is very likely, and if you can find that path then you can often find a synthesis between the two positions that is better than both.

Why Proof of Stake (Nov 2020)

2020 Nov 06

There are three key reasons why PoS is a superior blockchain security mechanism compared to PoW.

PoS offers more security for the same cost

The easiest way to see this is to put proof of stake and proof of work side by side, and **look at how much it costs to attack a network per $1 per day in block rewards**.

GPU-based proof of work

You can rent GPUs cheaply, so the cost of attacking the network is simply the cost of renting enough GPU power to outrun the existing miners. For every $1 of block rewards, the existing miners should be spending close to $1 in costs (if they're spending more, miners will drop out due to being unprofitable, if they're spending less, new miners can join in and take high profits). Hence, attacking the network just requires temporarily spending more than $1 per day, and only for a few hours.

Total cost of attack: ~$0.26 (assuming 6-hour attack), potentially reduced to zero as the attacker receives block rewards

ASIC-based proof of work

ASICs are a capital cost: you buy an ASIC once and you can expect it to be useful for ~2 years before it wears out and/or is obsoleted by newer and better hardware. If a chain gets 51% attacked, the community will likely respond by changing the PoW algorithm and your ASIC will lose its value. On average, mining is ~1/3 ongoing costs and ~2/3 capital costs (see here for some sources). Hence, per $1 per day in reward, miners will be spending ~$0.33 per day on electricity+maintenance and ~$0.67 per day on their ASIC. Assuming an ASIC lasts ~2 years, that's $486.67 that a miner would need to spend on that quantity of ASIC hardware.

Total cost of attack: $486.67 (ASICs) + $0.08 (electricity+maintenance) = $486.75

That said, it's worth noting that ASICs provide this heightened level of security against attacks at a high cost of centralization, as the barriers to entry to joining become very high.

Proof of stake

Proof of stake is almost entirely capital costs (the coins being deposited); the only operating costs are the cost of running a node. Now, how much capital are people willing to lock up to get $1 per day of rewards? **Unlike ASICs, deposited coins do not depreciate, and when you're done staking you get your coins back after a short delay. Hence, participants should be willing to pay much higher capital costs for the same quantity of rewards**.

Let's assume that a ~15% rate of return is enough to motivate people to stake (that is the expected eth2 rate of return). Then, $1 per day of rewards will attract 6.667 years' worth of returns in deposits, or $2433. Hardware and electricity costs of a node are small; a thousand-dollar computer can stake for hundreds of thousands of dollars in deposits, and ~$100 per months in electricity and internet is sufficient for such an amount. But conservatively, we can say these ongoing costs are ~10% of the total cost of staking, so we only have $0.90 per day of rewards that end up corresponding to capital costs, so we do need to cut the above figure by ~10%.

Total cost of attack: $0.90/day * 6.667 years = $2189

In the long run, this cost is expected to go even higher, as staking becomes more efficient and people become comfortable with lower rates of return. I personally expect this number to eventually rise to something like $10000.

Note that the only "cost" being incurred to get this high level of security is just the inconvenience of not being able to move your coins around at will while you are staking. It may even be the case that

the public knowledge that all these coins are locked up causes the value of the coin to rise, so the total amount of money floating around in the community, ready to make productive investments etc, remains the same! Whereas **in PoW, the "cost" of maintaining consensus is real electricity being burned in [insanely large quantities](#).**

Higher security or lower costs?

Note that there are two ways to use this 5-20x gain in security-per-cost. One is to keep block rewards the same but benefit from increased security. The other is to massively reduce block rewards (and hence the "waste" of the consensus mechanism) and keep the security level the same.

Either way is okay. I personally prefer the latter, because as we will see below, in proof of stake even a successful attack is much less harmful and much easier to recover from than an attack on proof of work!

Attacks are much easier to recover from in proof of stake

In a proof of work system, if your chain gets 51% attacked, what do you even do? So far, the only response in practice has been "wait it out until the attacker gets bored". But this misses the possibility of a much more dangerous kind of attack called a **spawn camping attack**, where the attacker attacks the chain over and over again with the explicit goal of rendering it useless.

In a GPU-based system, there is no defense, and a persistent attacker may quite easily render a chain permanently useless (or more realistically, switches to proof of stake or proof of authority). In fact, after the first few days, the attacker's costs may become very low, as honest miners will drop out since they have no way to get rewards while the attack is going on.

In an ASIC-based system, the community can respond to the first attack, but continuing the attack from there once again becomes trivial. The community would meet the first attack by hard-forking to change the PoW algorithm, thereby "bricking" all ASICs (the attacker's *and* honest miners'!). But if the attacker is willing to suffer that initial expense, after that point the situation reverts to the GPU case (as there is not enough time to build and distribute ASICs for the new algorithm), and so from there the attacker can cheaply continue the spawn camp inevitably.

In the PoS case, however, things are much brighter. For certain kinds of 51% attacks (particularly, reverting finalized blocks), there is a built-in "slashing" mechanism in [the proof of stake consensus](#) by which a large portion of the attacker's stake (and no one else's stake) can get automatically destroyed. For other, harder-to-detect attacks (notably, a 51% coalition censoring everyone else), the community can coordinate on a **minority user-activated soft fork (UASF)** in which the attacker's funds are once again largely destroyed (in Ethereum, this is done via the "inactivity leak mechanism"). **No explicit "hard fork to delete coins" is required; with the exception of the requirement to coordinate on the UASF to select a minority block, everything else is automated and simply following the execution of the protocol rules**.

Hence, attacking the chain the first time will cost the attacker many millions of dollars, and the community will be back on their feet within days. Attacking the chain the second time will still cost the attacker many millions of dollars, as they would need to buy new coins to replace their old coins that were burned. And the third time will... cost even more millions of dollars. **The game is very asymmetric, and not in the attacker's favor**.

Proof of stake is more decentralized than ASICs

GPU-based proof of work is reasonably decentralized; it is not too hard to get a GPU. But GPU-based mining largely fails on the "security against attacks" criterion that we mentioned above. ASIC-based mining, on the other hand, requires millions of dollars of capital to get into (and if you buy an ASIC from someone else, most of the time, the manufacturing company gets the far better end of the deal).

This is also the correct answer to the common "proof of stake means the rich get richer" argument: ASIC mining *also* means the rich get richer, and that game is *even more* tilted in favor of the rich. At least in PoS the minimum needed to stake is quite low and within reach of many regular people.

Additionally, proof of stake is more censorship resistant. GPU mining and ASIC mining are both very easy to detect: they require huge amounts of electricity consumption, expensive hardware purchases and large warehouses. PoS staking, on the other hand, can be done on an unassuming laptop and even over a VPN.

Possible advantages of proof of work

There are two primary genuine advantages of PoW that I see, though I see these advantages as being fairly limited.

Proof of stake is more like a "closed system", leading to higher wealth concentration over the long term

In proof of stake, if you have some coin you can stake that coin and get more of that coin. In proof of work, you can always earn more coins, but you need some outside resource to do so. Hence, one could argue that over the long term, proof of stake coin distributions risk becoming more and more concentrated.

The main response to this that I see is simply that in PoS, the rewards in general (and hence validator revenues) will be quite low; in eth2, we are expecting annual validator rewards to equal ~0.5-2% of the total ETH supply. And the more validators are staking, the lower interest rates get. Hence, it would likely take over a century for the level of concentration to double, and on such time scales other pressures (people wanting to spend their money, distributing their money to charity or among their children, etc.) are likely to dominate.

Proof of stake requires "weak subjectivity", proof of work does not

See [here](#) for the original intro to the concept of "weak subjectivity". Essentially, the first time a node comes online, and any subsequent time a node comes online after being offline for a very long duration (ie. multiple months), that node must find some third-party source to determine the correct head of the chain. This could be their friend, it could be exchanges and block explorer sites, the client developers themselves, or many other actors. PoW does not have this requirement.

However, arguably this is a very weak requirement; in fact, users need to trust client developers and/or "the community" to about this extent already. At the very least, users need to trust someone (usually client developers) to tell them what the protocol is and what any updates to the protocol have been. This is unavoidable in any software application. Hence, the marginal additional trust requirement that PoS imposes is still quite low.

But even if these risks do turn out to be significant, they seem to me to be much lower than the immense gains that PoS sytems get from their far greater efficiency and their better ability to handle and recover from attacks.

See also: my previous pieces on proof of stake.

- [Proof of Stake FAQ](#)
- [A Proof of Stake Design Philosophy](#)

Gitcoin Grants Round 7 Retrospective

2020 Oct 18

See all posts

Round 7 of Gitcoin Grants has successfully completed! This round has seen an unprecedented growth in interest and contributions, with $274,830 in contributions and $450,000 in matched funds distributed across 857 projects.

The category structure was once again changed; this time was had a split between "dapp tech", "infrastructure tech" and "community". Here are the results:

Defi joins the matching!

In this round, we were able to have much higher matching values than before. This was because the usual matchings, provided by the Ethereum Foundation and a few other actors, were supplemented for the first time by a high level of participation from various defi projects:

The matchers were:

- [Chainlink](#), a smart contract oracle project
- [Optimism](#), a layer-2 optimistic rollup
- The [Ethereum Foundation](#)
- [Balancer](#), a decentralized exchange
- [Synthetix](#), a synthetic assets platform
- [Yearn](#), a collateralized-lending platform
- [Three Arrows Capital](#), an investment fund
- [Defiance Capital](#), another investment fund
- [Future Fund](#), which is totally not an investment fund! (/s)
- $MEME, a memecoin
- [Yam](#), a defi project
- Some individual contributors: ferretpatrol, [bantg](#), [Mariano Conti](#), [Robert Leshner](#), [Eric Conner](#), [10b576da0](#)

The projects together contributed a large amount of matching funding, some of which was used this round and some of which is reserved as a "rainy

day fund" for future rounds in case future matchers are less forthcoming.

This is a significant milestone for the ecosystem because it shows that Gitcoin Grants is expanding beyond reliance on a very small number of funders, and is moving toward something more sustainable. But it is worth exploring, what exactly is driving these matchers to contribute, and is it sustainable?

There are a few possibile motivations that are likely all in play to various extents:

1. People are naturally altruistic to some extent, and this round defi projects got unexpectedly wealthy for the first time due to a rapid rise in interest and token prices, and so donating some of that windfall felt like a natural "good thing to do"
2. Many in the community are critical of defi projects by default, viewing them as unproductive casinos that create a negative image of what Ethereum is supposed to be about. Contributing to public goods is an easy way for a defi project to show that they want to be a positive contributor to the ecosystem and make it better
3. Even in the absence of such negative perceptions, defi is a competitive market that is heavily dependent on community support and network effects, and so it's very valuable to a project to win friends in the ecosystem
4. The largest defi projects capture enough of the benefit from these public goods that it's in their own interest to contribute
5. There's a high degree of common-ownership between defi projects (holders of one token also hold other tokens and hold ETH), and so even if it's not strictly in a *project*'s interest to donate a large amount, *token holders of that project* push the project to contribute because they as holders benefit from the gains to both that project but also to the other projects whose tokens they hold.

The remaining question is, of course: how sustainable will these incentives be? Are the altruistic and public-relations incentives only large enough for a one-time burst of donations of this size, or could it become more sustainable? Could we reliably expect to see, say, $2-3 million per year spent on quadratic funding matching from here on? If so, it would be excellent news for public goods funding diversification and democratization in the Ethereum ecosystem.

Where did the troublemakers go?

One curious result from the previous round and this round is that the "controversial" community grant recipients from previous rounds seem to have dropped in prominence on their own. In theory, we should have seen them continue to get support from their supporters with their detractors being able to do nothing about it. In practice, though, the top media recipients this round appear to be relatively uncontroversial and universally beloved mainstays of the Ethereum ecosystem. Even the Zero Knowledge Podcast, an excellent podcast but one aimed for a relatively smaller and more highly technical audience, has received a large contribution this round.

What happened? Why did the distribution of media recipients improve in quality all on its own? Is the mechanism perhaps more self-correcting than we had thought?

Overpayment

This round is the first round where top recipients on all sides received quite a large amount. On the infrastructure side, the White Hat Hacking project (basically a fund to donate to samczsun) received a total of $39,258, and the Bankless podcast got $47,620. We could ask the question: are the top recipients getting too much funding?

To be clear, I do think that it's very improper to try to create a moral norm that public goods contributors should only be earning salaries up to a certain level and should not be able to earn much more than that. People launching coins earn huge windfalls all the time; it is completely natural and fair for public goods contributors to also get that possibility (and furthermore, the numbers from this round translate to about ~$200,000 per year, which is not even that high).

However, one can ask a more limited and pragmatic question: given the current reward structure, is putting an extra $1 into the hands of a top contributor *less valuable* than putting $1 into the hands of one of the other very valuable projects that's still underfunded? Turbogeth, Nethermind, RadicalXChange and many other projects could still do quite a lot with a marginal dollar. For the first time, the matching amounts are high enough that this is actually a significant issue.

Especially if matching amounts increase even further, is the ecosystem going to be able to correctly allocate funds and avoid overfunding projects? Alternatively, if it fails to avoid over-concentrating funds, is that all that bad? Perhaps the possibility of becoming the center of attention for one round and earning a $500,000 windfall will be part of the incentive that motivates independent public goods contributors!

We don't know; but these are the yet-unknown facts that running the experiment at its new increased scale is for the first time going to reveal.

Let's talk about categories...

The concept of categories as it is currently implemented in Gitcoin Grants is a somewhat strange one. Each category has a fixed total matching amount that is split between projects within that category. What this mechanism basically says is that the community can be trusted to choose between projects *within* a category, but we need a separate technocratic judgement to judge how the funds are split between the different categories in the first place.

But it gets more paradoxical from here. In Round 7, a "collections" feature was introduced halfway through the round:

Ethereum State Media

owocki . 12 grants

A collection of grants that are putting out the good word about Ethereum + fighting disinformation.

Add to Cart

Share Collection

Sassal's Cart of Love

sassal . 42 grants

Add to Cart

Share Collection

Blockchain Education & Community

1stcryptoking . 24 grants

Add to Cart

Share Collection

DeFi Locals

lemogras . 8 grants

If you want global adaptation, please support local/Non-English source of informations.

Add to Cart

Share Collection

Full Self-Sovereign Life

tze42 . 9 grants

This short collection combines tools that are needed to share value among your peers in a fully decentralized fashion.

Add to Cart

Share Collection

Privacy

ceresstation . 7 grants

Privacy isn't needed only when you have something to hide in the same way that free speech isn't needed only when you have something to say

Add to Cart

Share Collection

If you click "Add to Cart" on a collection, you immediately add everything in the collection to your cart. This is strange because this mechanism seems to send the exact *opposite* message: users that don't understand the details well can choose to allocate funds to entire categories, but (unless they manually edit the amounts) they should not be making many active decisions *within* each category.

Which is it? Do we trust the radical quadratic fancy democracy to allocate within categories but not between them, do we trust it to allocate between categories but nudge people away from making fine-grained decisions within them, or something else entirely? I recommend that for Round 8 we think harder about the philosophical challenges here and come up with a more principled approach.

One option would be to have one matching pool and have *all* the categories just be a voluntary UI layer. Another would be to experiment with *even more* "affirmative action" to bootstrap particular categories: for example, we could split the Community matching into a $25,000 matching pool for each major world region (eg. North America + Oceania, Latin America, Europe, Africa, Middle East, India, East + Southeast Asia) to try to give projects in more neglected areas a leg up. There are many possibilities here! One hybrid route is that the "focused" pools could *themselves* be quadratic funded in the previous round!

Identity verification

As collusion, fake accounts and other attacks on Gitcoin Grants have been recently increasing, Round 7 added an additional verification option with the decentralized social-graph-based [BrightID](#), and single-handedly boosted the project's userbase by a factor of ten:

BrightID continues to grow like crazy thanks to the trajectory Gitcoin helped put us on.
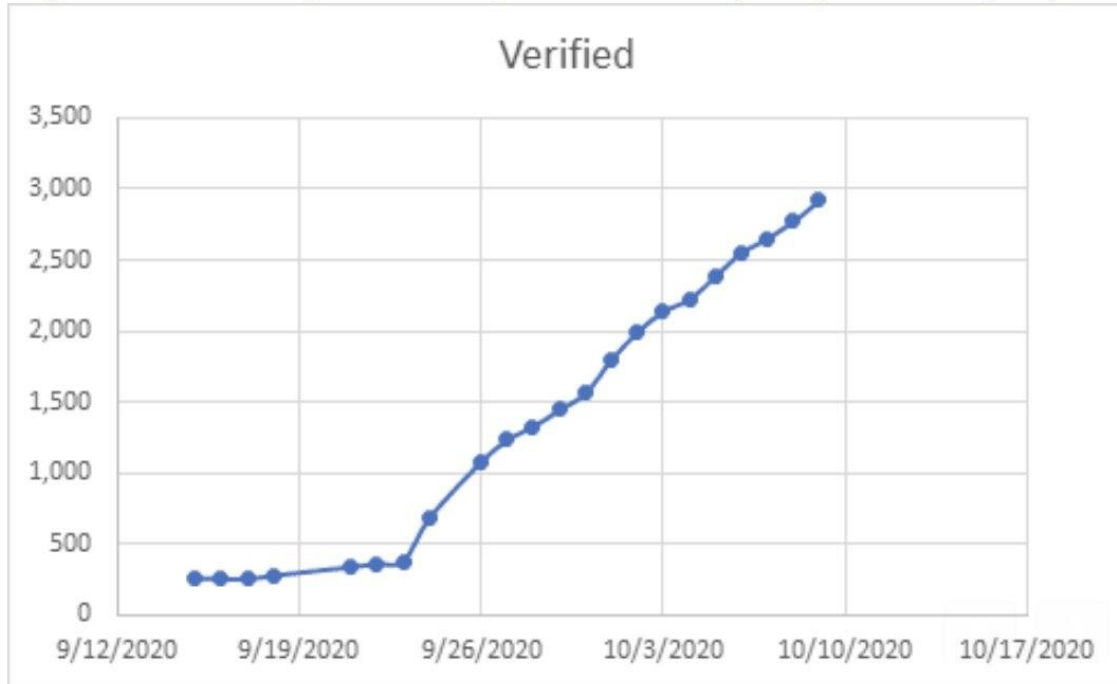
This is good, because along with helping BrightID's growth, it also subjects the project to a trial-by-fire: there's now a large incentive to try to create a large number of fake accounts on it! BrightID is going to face a tough challenge making it reasonably easy for regular users to join but at the same time resist attacks from fake and duplicate accounts. I look forward to seeing them try to meet the challenge!

ZK rollups for scalability

Finally, Round 7 was the first round where Gitcoin Grants experimented with using the ZkSync ZK rollup to decrease fees for payments:
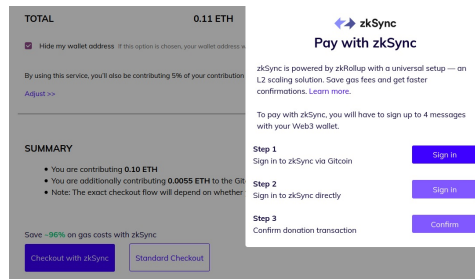
The main thing to report here is simply that the ZK rollup successfully did decrease fees! The user experience worked well. Many optimistic and ZK rollup projects are now looking at collaborating with wallets on direct integrations, which should increase the usability and security of such techniques further.

Conclusions

Round 7 has been a pivotal round for Gitcoin Grants. The matching funding has become much more sustainable. The levels of funding are now large enough to successfully fund quadratic freelancers to the point where a project getting "too much funding" is a conceivable thing to worry about! Identity verification is taking steps forward. Payments have become much more efficient with the introduction of the ZkSync ZK rollup. I look forward to seeing the grants continue for many more rounds in the future.

Coordination, Good and Bad

Special thanks to Karl Floersch and Jinglan Wang for feedback and review

See also:

- [On Collusion](#)
- [Engineering Security Through Coordination Problems](#)
- [Trust Models](#)
- [The Meaning Of Decentralization](#)

Coordination, the ability for large groups of actors to work together for their common interest, is one of the most powerful forces in the universe. It is the difference between a king comfortably ruling a country as an oppressive dictatorship, and the people coming together and overthrowing him. It is the difference between the global temperature going up [3-5'C](#) and the temperature going up by a much smaller amount if we work together to stop it. And it is the factor that makes companies, countries and any social organization larger than a few people possible at all.

Coordination can be improved in many ways: faster spread of information, better norms that identify what behaviors are classified as cheating along with more effective punishments, stronger and more powerful organizations, tools like smart contracts that allow interactions with reduced levels of trust, governance technologies (voting, shares, decision markets...), and much more. And indeed, we as a species are getting better at all of these things with each passing decade.

But there is also a very philosophically counterintuitive dark side to coordination. **While it is emphatically true that "everyone coordinating with everyone" leads to much better outcomes than "every man for himself", what that does NOT imply is that each individual step toward more coordination is necessarily beneficial**. If coordination is improved in an unbalanced way, the results can easily be harmful.

We can think about this visually as a map, though in reality the map has many billions of "dimensions" rather than two:
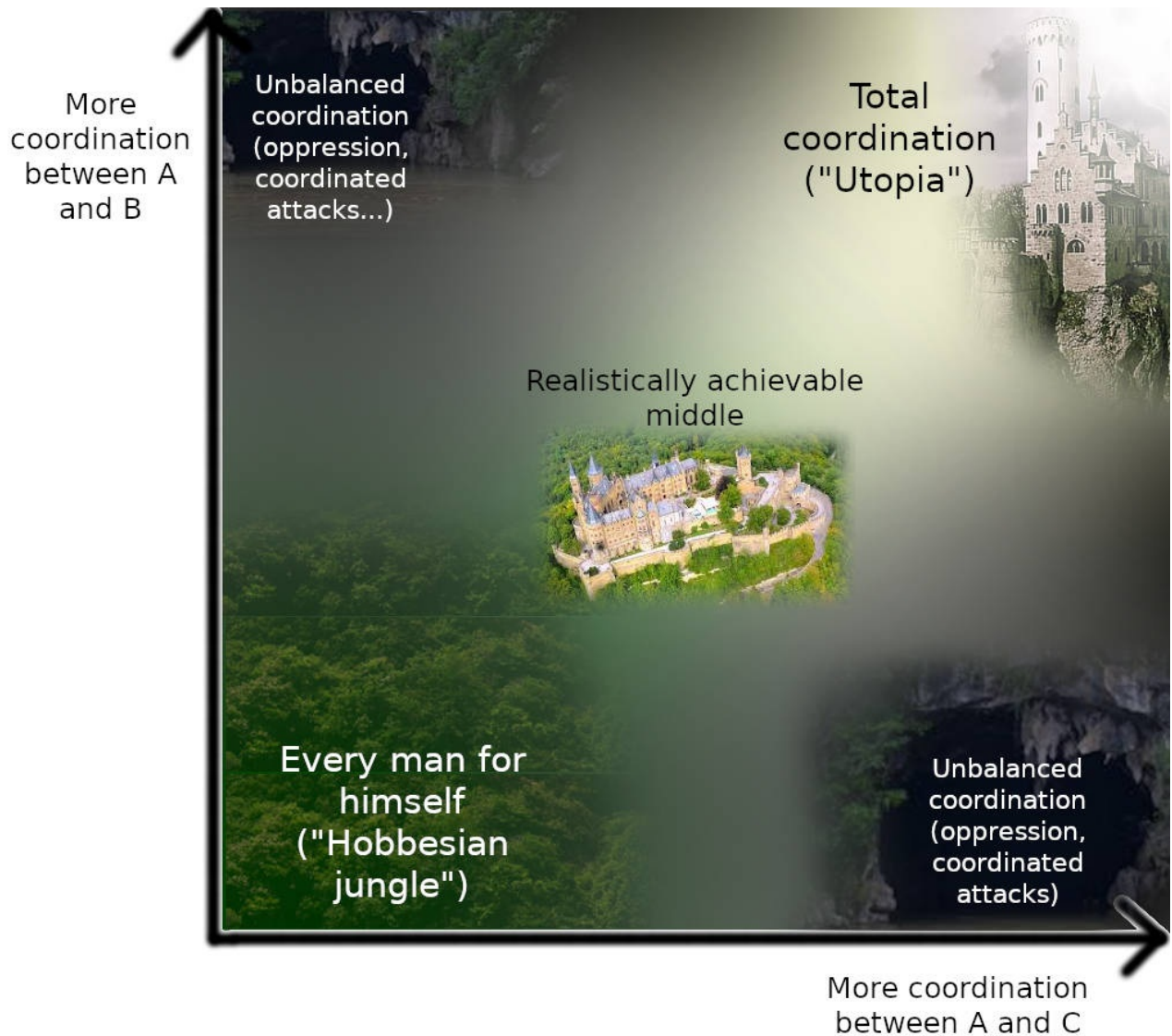
The bottom-left corner, "every man for himself", is where we don't want to be. The top-right corner, total coordination, is ideal, but likely unachievable. But the landscape in the middle is far from an even slope up, with many reasonably safe and productive places that it might be best to settle down in and many deep dark caves to avoid.

Now what are these dangerous forms of partial coordination, where someone coordinating with *some* fellow humans but not *others* leads to a deep dark hole? It's best to describe them by giving examples:

- Citizens of a nation valiantly sacrificing themselves for the greater good of their country in a war.... when that country turns out to be WW2-era Germany or Japan
- A lobbyist giving a politician a bribe in exchange for that politician adopting the lobbyist's preferred policies
- Someone selling their vote in an election
- All sellers of a product in a market colluding to raise their prices at the same time
- Large miners of a blockchain colluding to launch a 51% attack

In all of the above cases, we see a group of people coming together and cooperating with each other, but to the great detriment of some group that is outside the circle of coordination, and thus to the net detriment of the world as a whole. In the first case, it's all the people that were the victims of the aforementioned nations' aggression that are outside the circle of coordination and suffer heavily as a result; in the second and third cases, it's the people affected by the decisions that the corrupted voter and politician are making, in the fourth case it's the customers, and in the fifth case it's the non-participating miners and the blockchain's users. It's not an individual defecting against the group, it's a group defecting against a broader group, often the world as a whole.

This type of partial coordination is often called "collusion", but it's important to note that the range of behaviors that we are talking about is quite broad. In normal speech, the word "collusion" tends to be used more often to describe relatively symmetrical relationships, but in the above cases there are plenty of examples with a strong asymmetric character. Even *extortionate* relationships ("vote for my preferred policies or I'll publicly reveal your affair") are a form of collusion in this sense. In the rest of this post, we'll use "collusion" to refer to "undesired coordination" generally.

Evaluate Intentions, Not Actions (!!)

One important property of especially the milder cases of collusion is that one cannot determine whether or not an action is part of an undesired collusion just by looking at the action itself. The reason is that the actions that a person takes are a combination of that person's internal knowledge, goals and preferences together with externally imposed incentives on that person, and so the actions that people take when colluding, versus the actions that people take on their own volition (or coordinating in benign ways) often overlap.

For example, consider the case of collusion between sellers (a type of antitrust violation). If operating independently, each of three sellers might set a price for some product between $5 and $10; the differences within the range reflect difficult-to-see factors such as the seller's internal costs, their own willingness to work at different wages, supply-chain issues and the like. But if the sellers collude, they might set a price between $8 and $13. Once again, the range reflects different possibilities regarding internal costs and other difficult-to-see factors. If you see someone selling that product for $8.75, are they doing something wrong? Without knowing whether or not they coordinated with other sellers, you can't tell! Making a law that says that selling that product for more than $8 would be a bad idea; maybe there are legitimate reasons why prices have to be high at the current time. But making a law against collusion, and successfully enforcing it, gives the ideal outcome - you get the $8.75 price if the price has to be that high to cover sellers' costs, but you don't get that price if the factors driving prices up naturally are low.

This applies in the bribery and vote selling cases too: it may well be the case that some people vote for the Orange Party legitimately, but others vote for the Orange Party because they were paid to. From the point of view of someone determining the rules for the voting mechanism, they don't know ahead of time whether the Orange Party is good or bad. But what they *do* know is that a vote where people vote based on their honest internal feelings works reasonably well, but a vote where voters can freely buy and sell their votes works terribly. This is because vote selling has a tragedy-of-the-commons: each voter only gains a small portion of the benefit from voting correctly, but would gain the full bribe if they vote the way the briber wants, and so the required bribe to lure each individual voter is far smaller than the bribe that would actually compensate the population for the costs of whatever policy the briber wants. Hence, votes where vote selling is permitted quickly collapse into plutocracy.

Understanding the Game Theory

We can zoom further out and look at this from the perspective of game theory. In the version of game theory that focuses on individual choice - that is, the version that assumes that each participant makes decisions independently and that does not allow for the possibility of groups of agents working as one for their mutual benefit, there are mathematical proofs that at least one stable Nash equilibrium must exist in any game. In fact, mechanism designers have a very wide latitude to "engineer" games to achieve specific outcomes. But in the version of game theory that allows for the possibility of coalitions working together (ie. "colluding"), called *cooperative game theory*, we can prove that there are large classes of games that do not have any stable outcome (called a "core"). In such games, whatever the current state of affairs is, there is always some coalition that can profitably deviate from it.

One important part of that set of inherently unstable games is *majority games*. A majority game is formally described as a game of agents where any subset of more than half of them can capture a fixed reward and split it among themselves - a setup eerily similar to many situations in corporate governance, politics and many other situations in human life. That is to say, if there is a situation with some fixed pool of resources and some currently established mechanism for distributing those resources, and it's unavoidably possible for 51% of the participants can conspire to seize control of the resources, no matter what the current configuration is there is always some conspiracy that can emerge that would be profitable for the participants. However, that conspiracy would then in turn be vulnerable to potential new conspiracies, possibly including a combination of previous conspirators and victims... and so on and so forth.

Round	A	B	C
1	1/3	1/3	1/3
2	1/2	1/2	0
3	2/3	0	1/3
4	0	1/3	2/3

This fact, the instability of majority games under cooperative game theory, is arguably highly underrated as a simplified general mathematical model of why there may well be no "end of history" in politics and no system that proves fully satisfactory; I personally believe it's much more useful than the more famous Arrow's theorem, for example.

Note once again that the core dichotomy here is not "individual versus group"; for a mechanism designer, "individual versus group" is surprisingly easy to handle. It's "group versus broader group" that presents the challenge.

Decentralization as Anti-Collusion

But there is another, brighter and more actionable, conclusion from this line of thinking: if we want to create mechanisms that are stable, then we know that one important ingredient in doing so is finding ways to make it more difficult for collusions, especially large-scale collusions, to happen and to maintain themselves. In the case of voting, we have the secret ballot - a mechanism that ensures that voters have no way to prove to third parties how they voted, even if they want to prove it (MACI is one project trying to use cryptography to extend secret-ballot principles to an online context). This disrupts trust between voters and bribers, heavily restricting undesired collusions that can happen. In that case of antitrust and other corporate malfeasance, we often rely on whistleblowers and even give them rewards, explicitly incentivizing participants in a harmful collusion to defect. And in the case of public infrastructure more broadly, we have that oh-so-important concept: **decentralization**.

One naive view of why decentralization is valuable is that it's about reducing risk from single points of technical failure. In traditional "enterprise" distributed systems, this is often actually true, but in many other cases we know that this is not sufficient to explain what's going on. It's instructive here to look at blockchains. A large mining pool publicly showing how they have internally distributed their nodes and network dependencies doesn't do much to calm community members scared of mining centralization. And pictures like these, showing 90% of Bitcoin hashpower at the time being capable of showing up to the same conference panel, do quite a bit to scare people:

But why is this image scary? From a "decentralization as fault tolerance" view, large miners being able to talk to each other causes no harm. But if we look at "decentralization" as being the presence of barriers to harmful collusion, then the picture becomes quite scary, because it shows that those barriers are not nearly as strong as we thought. Now, in reality, the barriers are still far from zero; the fact that those miners can easily perform technical coordination and likely are all in the same Wechat groups does *not*, in fact, mean that Bitcoin is "in practice little better than a centralized company".

So what are the remaining barriers to collusion? Some major ones include:

- **Moral Barriers**. In *Liars and Outliers*, Bruce Schneier reminds us that many "security systems" (locks on doors, warning signs reminding people of punishments...) also serve a moral function, reminding potential misbehavers that they are about to conduct a serious transgression and if they want to be a good person they should not do that. Decentralization arguably serves that function.
- **Internal negotiation failure**. The individual companies may start demanding concessions in exchange for participating in the collusion, and this could lead to negotiation stalling outright (see "holdout problems" in economics).
- **Counter-coordination**. The fact that a system is decentralized makes it easy for participants not participating in the collusion to make a fork that strips out the colluding attackers and continue the system from there. Barriers for users to join the fork are low, and the *intention* of decentralization creates moral pressure in favor of participating in the fork.
- **Risk of defection**. It still is much harder for five companies to join together to do something widely considered to be bad than it is for them to join together for a non-controversial or benign purpose. The five companies do not know each other *too* well, so there is a risk that one of them will refuse to participate and blow the whistle quickly, and the participants have a hard time judging the risk. Individual employees within the companies may blow the whistle too.

Taken together, these barriers are substantial indeed - often substantial enough to stop potential attacks in their tracks, even when those five companies are simultaneously perfectly capable of quickly coordinating to do something legitimate. Ethereum blockchain miners, for example, are perfectly capable of coordinating increases to the gas limit, but that does not mean that they can so easily collude to attack the chain.

The blockchain experience shows how designing protocols as institutionally decentralized architectures, even when it's well-known ahead of time that the bulk of the activity will be dominated by a few companies, can often be a very valuable thing. This idea is not limited to blockchains; it can be applied in other contexts as well (eg. see here for applications to antitrust).

Forking as Counter-Coordination

But we cannot always effectively prevent harmful collusions from taking place. And to handle those cases where a harmful collusion does take place, it would be nice to make systems that are more robust against them - more expensive for those colluding, and easier to recover for the system.

There are two core operating principles that we can use to achieve this end: (1) **supporting counter-coordination** and (2) **skin-in-the-game**. The idea behind counter-coordination is this: we know that we cannot design systems to be *passively* robust to collusions, in large part because there is an extremely large number of ways to organize a collusion and there is no passive mechanism that can detect them, but what we can do is *actively* respond to collusions and strike back.

In digital systems such as blockchains (this could also be applied to more mainstream systems, eg. DNS), a major and crucially important form of counter-coordination is **forking**.

If a system gets taken over by a harmful coalition, the dissidents can come together and create an alternative version of the system, which has (mostly) the same rules except that it removes the power of the attacking coalition to control the system. Forking is very easy in an open-source software context; the main challenge in creating a successful fork is usually gathering the **legitimacy** (game-theoretically viewed as a form of "common knowledge") needed to get all those who disagree with the main coalition's direction to follow along with you.

This is not just theory; it has been accomplished successfully, most notably in the Steem community's rebellion against a hostile takeover attempt, leading to a new blockchain called Hive in which the original antagonists have no power.

Markets and Skin in the Game

Another class of collusion-resistance strategy is the idea of **skin in the game**. Skin in the game, in this context, basically means any mechanism that holds individual contributors in a decision individually accountable for their contributions. If a group makes a bad decision, those who approved the decision must suffer more than those who attempted to dissent. This avoids the "tragedy of the commons" inherent in voting systems.

Forking is a powerful form of counter-coordination precisely because it introduces skin in the game. In Hive, the community fork of Steem that threw off the hostile takeover attempt, the coins that were used to vote in favor of the hostile takeover were largely deleted in the new fork. The key individuals who participated in the attack individually suffered as a result.

Markets are in general very powerful tools precisely because they maximize skin in the game. **Decision markets** (prediction markets used to guide decisions; also called futarchy) are an attempt to extend this benefit of markets to organizational decision-making. That said, decision markets can only solve some problems; in particular, they cannot tell us what variables we should be optimizing for in the first place.

Structuring Coordination

This all leads us to an interesting view of what it is that people building social systems *do*. One of the goals of building an effective social system is, in large part, determining *the structure of coordination*: which groups of people and in what configurations can come together to further their group goals, and which groups cannot?
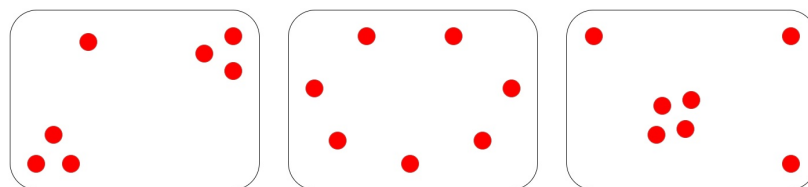
Different coordination structures, different outcomes

Sometimes, more coordination is good: it's better when people can work together to collectively solve their problems. At other times, more coordination is dangerous: a subset of participants could coordinate to disenfranchise everyone else. And at still other times, more coordination is necessary for another reason: to enable the broader community to "strike back" against a collusion attacking the system.

In all three of those cases, there are different mechanisms that can be used to achieve these ends. Of course, it is very difficult to prevent communication outright, and it is very difficult to make coordination perfect. But there are many options in between that can nevertheless have powerful effects.

Here are a few possible coordination structuring techniques:

- Technologies and norms that protect privacy

- Technological means that make it difficult to prove how you behaved (secret ballots, MACI and similar tech)
- Deliberate decentralization, distributing control of some mechanism to a wide group of people that are known to not be well-coordinated
- Decentralization in physical space, separating out different functions (or different shares of the same function) to different locations (eg. see [Samo Burja on connections between urban decentralization and political decentralization](#))
- Decentralization between role-based constituencies, separating out different functions (or different shares of the same function) to different types of participants (eg. in a blockchain: "core developers", "miners", "coin holders", "application developers", "users")
- [Schelling points](#), allowing large groups of people to quickly coordinate around a single path forward. Complex Schelling points could potentially even be implemented in code (eg. [recovery from 51% attacks](#) can benefit from this).
- Speaking a common language (or alternatively, splitting control between multiple constituencies who speak different languages)
- Using per-person voting instead of per-(coin/share) voting to greatly increase the number of people who would need to collude to affect a decision
- Encouraging and relying on defectors to alert the public about upcoming collusions

None of these strategies are perfect, but they can be used in various contexts with differing levels of success. Additionally, these techniques can and should be combined with mechanism design that attempts to make harmful collusions less profitable and more risky to the extent possible; skin in the game is a very powerful tool in this regard. Which combination works best ultimately depends on your specific use case.

Trust Models

2020 Aug 20

One of the most valuable properties of many blockchain applications is *trustlessness*: the ability of the application to continue operating in an expected way without needing to rely on a specific actor to behave in a specific way even when their interests might change and push them to act in some different unexpected way in the future. Blockchain applications are never *fully* trustless, but some applications are much closer to being trustless than others. If we want to make practical moves toward trust minimization, we want to have the ability to compare different degrees of trust.

First, my simple one-sentence definition of trust: **trust is the use of any assumptions about the behavior of other people**. If before the pandemic you would walk down the street without making sure to keep two meters' distance from strangers so that they could not suddenly take out a knife and stab you, that's a kind of trust: both trust that people are very rarely completely deranged, and trust that the people managing the legal system continue to provide strong incentives against that kind of behavior. When you run a piece of code written by someone else, you trust that they wrote the code honestly (whether due to their own sense of decency or due to an economic interest in maintaining their reputations), or at least that *there exist* enough people checking the code that a bug would be found. Not growing your own food is another kind of trust: trust that enough people will realize that it's in *their* interests to grow food so they can sell it to you. You can trust different sizes of groups of people, and there are different kinds of trust.

For the purposes of analyzing blockchain protocols, I tend to break down trust into four dimensions:

- How many people do you need to behave as you expect?
- Out of how many?
- What kinds of motivations are needed for those people to behave? Do they need to be altruistic, or just profit seeking? Do they need to be uncoordinated?
- How badly will the system fail if the assumptions are violated?

For now, let us focus on the first two. We can draw a graph:
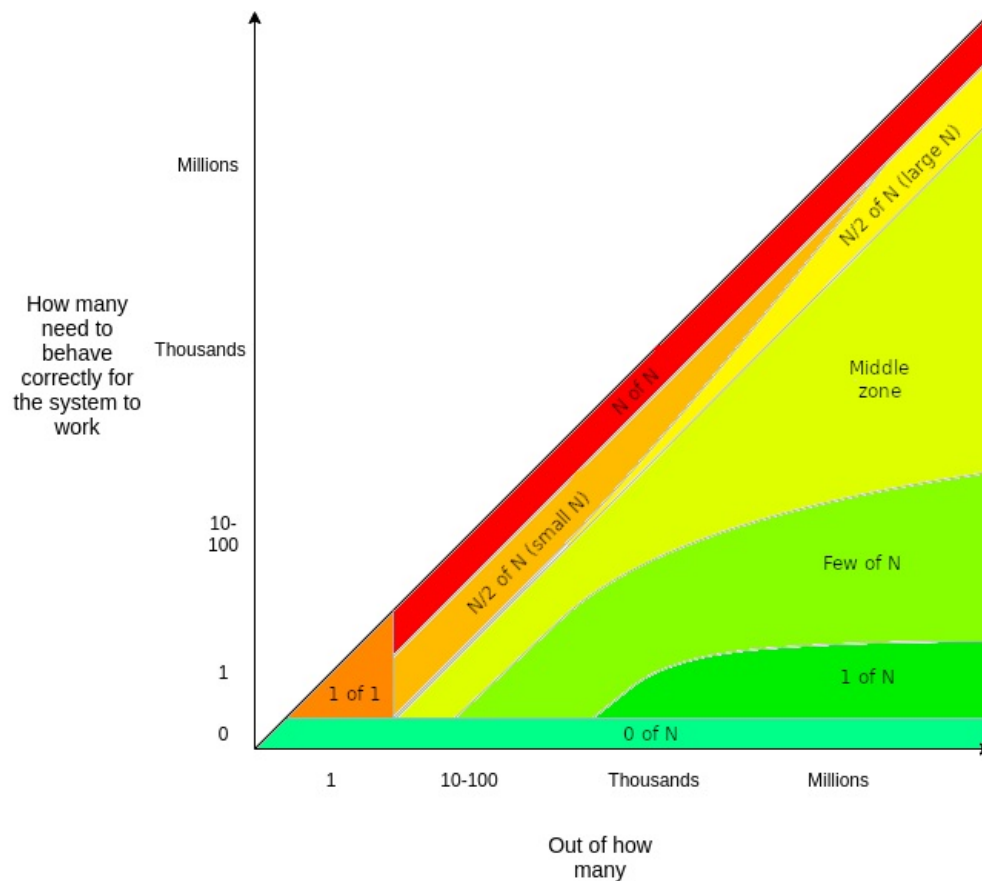
The more green, the better. Let us explore the categories in more detail:

- **1 of 1**: there is exactly one actor, and the system works if (and only if) that one actor does what you expect them to. This is the traditional "centralized" model, and it is what we are trying to do better than.
- **N of N**: the "dystopian" world. You rely on a whole bunch of actors, *all* of whom need to act as expected for everything to work, with no backups if any of them fail.
- **N/2 of N**: this is how blockchains work - they work if the majority of the miners (or PoS validators) are honest. Notice that N/2 of N becomes significantly more valuable the larger the N gets; a blockchain with a few miners/validators dominating the network is much less interesting than a blockchain with its miners/validators widely distributed. That said, we want to improve on even this level of security, hence the [concern around surviving 51% attacks](#).
- **1 of N**: there are many actors, and the system works as long as at least one of them does what you expect them to. Any system based on fraud proofs falls into this category, as do trusted setups though in that case the N is often smaller. Note that you do want the N to be as large as possible!
- **Few of N**: there are many actors, and the system works as long as at least some small fixed number of them do what you expect them do. [Data availability checks](#) fall into this category.
- **0 of N**: the systems works as expected without any dependence whatsoever on external actors. Validating a block by checking it yourself falls into this category.

While all buckets other than "0 of N" can be considered "trust", they are very different from each other! Trusting that one particular person (or organization) will work as expected is very different from trusting that *some single person anywhere* will do what you expect them to. "1 of N" is arguably much closer to "0 of N" than it is to "N/2 of N" or "1 of 1". A 1-of-N model might perhaps feel like a 1-of-1 model because it feels like you're going through a single actor, but the reality of the two is *very* different: in a 1-of-N system, if the actor you're working with at the moment disappears or turns evil, you can just switch to another one, whereas in a 1-of-1 system you're screwed.

Particularly, note that even the correctness of the software you're running typically depends on a "few of N" trust model to ensure that if there's bugs in the code someone will catch them. With that fact in mind, trying really hard to go from 1 of N to 0 of N on some other aspect of an application is often like making a reinforced steel door for your house when the windows are open.

Another important distinction is: how does the system fail if your trust assumption is violated? In blockchains, two most common types of failure are **liveness failure** and **safety failure**. A liveness failure is an event in which you are temporarily unable to do something you want to do (eg. withdraw coins, get a transaction included in a block, read information from the blockchain). A safety failure is an event in which something actively happens that the system was meant to prevent (eg. an invalid block gets included in a blockchain).

Here are a few examples of trust models of a few blockchain layer 2 protocols. I use "**small N**" to refer to the set of participants of the layer 2 system itself, and "**big N**" to refer to the participants of the blockchain; the assumption is always that the layer 2 protocol has a smaller community than the blockchain itself. I also limit my use of the word "liveness failure" to cases where coins are stuck for a significant amount of time; no longer being able to use the system but being able to near-instantly withdraw does not count as a liveness failure.

- **Channels** (incl state channels, lightning network): 1 of 1 trust for liveness (your counterparty can temporarily freeze your funds, though the harms of this can be mitigated if you split coins between multiple counterparties), N/2 of big-N trust for safety (a blockchain 51% attack can steal your coins)
- **Plasma** (assuming centralized operator): 1 of 1 trust for liveness (the operator can temporarily freeze your funds), N/2 of big-N trust for safety (blockchain 51% attack)
- **Plasma** (assuming semi-decentralized operator, eg. DPOS): N/2 of small-N trust for liveness, N/2 of big-N trust for safety
- **Optimistic rollup**: 1 of 1 or N/2 of small-N trust for liveness (depends on operator type), N/2 of big-N trust for safety
- **ZK rollup**: 1 of small-N trust for liveness (if the operator fails to include your transaction, you can withdraw, and if the operator fails to include your withdrawal immediately they cannot produce more batches and you can self-withdraw with the help of any full node of the rollup system); no safety failure risks
- **ZK rollup** (with [light-withdrawal enhancement](#)): no liveness failure risks, no safety failure risks

Finally, there is the question of incentives: does the actor you're trusting need to be very altruistic to

act as expected, only slightly altruistic, or is being rational enough? Searching for fraud proofs is "by default" slightly altruistic, though just how altruistic it is depends on the complexity of the computation (see the verifier's dilemma), and there are ways to modify the game to make it rational.

Assisting others with withdrawing from a ZK rollup is rational if we add a way to micro-pay for the service, so there is *really* little cause for concern that you won't be able to exit from a rollup with any significant use. Meanwhile, the greater risks of the other systems can be alleviated if we agree as a community to not accept 51% attack chains that revert too far in history or censor blocks for too long.

Conclusion: when someone says that a system "depends on trust", ask them in more detail what they mean! Do they mean 1 of 1, or 1 of N, or N/2 of N? Are they demanding these participants be altruistic or just rational? If altruistic, is it a tiny expense or a huge expense? And what if the assumption is violated - do you just need to wait a few hours or days, or do you have assets that are stuck forever? Depending on the answers, your own answer to whether or not you want to use that system might be very different.

A Philosophy of Blockchain Validation

2020 Aug 17

See also:

- [A Proof of Stake Design Philosophy](#)
- [The Meaning of Decentralization](#)
- [Engineering Security through Coordination Problems](#)

One of the most powerful properties of a blockchain is the fact that every single part of the blockchain's execution can be independently validated. Even if a great majority of a blockchain's miners (or validators in PoS) get taken over by an attacker, if that attacker tries to push through invalid blocks, the network will simply reject them. Even those users that were not verifying blocks at that time can be (potentially automatically) warned by those who were, at which point they can check that the attacker's chain is invalid, and automatically reject it and coordinate on accepting a chain that follows the rules.

But how much validation do we actually need? Do we need a hundred independent validating nodes, a thousand? Do we need a culture where the average person in the world runs software that checks every transaction? It's these questions that are a challenge, and a very important challenge to resolve especially if we want to build blockchains with consensus mechanisms better than the single-chain "Nakamoto" proof of work that the blockchain space originally started with.

Why validate?
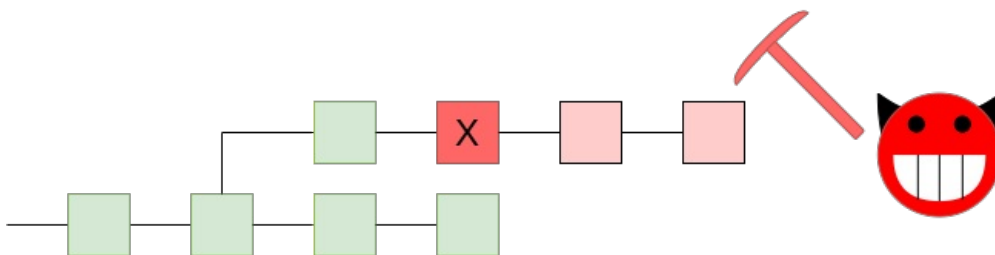
A 51% attack pushing through an invalid block. We want the network to reject the chain!

There are two main reasons why it's beneficial for a user to validate the chain. First, it maximizes the chance that the node can correctly determine and say on the **canonical chain** - the chain that the community accepts as legitimate. Typically, the canonical chain is defined as something like "the valid chain that has the most miners/validators supporting it" (eg. the "longest valid chain" in Bitcoin). Invalid chains are rejected by definition, and if there is a choice between multiple valid chains, the chain that has the most support from miners/validators wins out. And so if you have a node that verifies all the validity conditions, and hence detects which chains are valid and which chains are not, that maximizes your chances of correctly detecting what the canonical chain actually is.

But there is also another deeper reason why validating the chain is beneficial. Suppose that a powerful actor tries to push through a change to the protocol (eg. changing the issuance), and has the support of the majority of miners. If no one else validates the chain, this attack can very easily succeed: everyone's clients will, *by default*, accept the new chain, and by the time anyone sees what is going on, it will be *up to the dissenters* to try to coordinate a rejection of that chain. But if average users are validating, then the coordination problem falls on the other side: it's now the responsibility of whoever is trying to change the protocol to convince the users to actively download the software patch to accept the protocol change.

If enough users are validating, then **instead of defaulting to victory, a contentious attempt to force a change of the protocol will default to chaos**. Defaulting to chaos still causes a lot of

disruption, and would require out-of-band social coordination to resolve, but it places a much larger barrier in front of the attacker, and makes attackers much less confident that they will be able to get away with a clean victory, making them much less motivated to even try to start an attack. If *most* users are validating (directly or indirectly), and an attack has *only* the support of the majority of miners, then the attack will outright **default to failure** - the best outcome of all.

The definition view versus the coordination view

Note that this reasoning is very different from a different line of reasoning that we often hear: that a chain that changes the rules is somehow "by definition" not the correct chain, and that no matter how many other users accept some new set of rules, what matters is that you personally can stay on the chain with the old rules that you favor.

Here is one example of the "by definition" perspective from Gavin Andresen:

I'd like to propose this big-picture technical definition of "Bitcoin":

"Bitcoin" is the ledger of not-previously-spent, validly signed transactions contained in the chain of blocks that begins with the genesis block (hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f), follows the 21-million coin creation schedule, and has the most cumulative double-SHA256-proof-of-work.[1]

Here's another from the Wasabi wallet; this one comes even more directly from the perspective of explaining why full nodes are valuable:

When running a Bitcoin full node, you define the precise monetary rules that you voluntarily agree on. Nobody else forces this choice upon you. Thus any sovereign individual who wants to claim financial independence must run a full node. Once your own rules are firmly established, your software discovers other nodes in the Bitcoin peer-to-peer network which do not break your rules. These peers send you transactions and blocks which are valid according to their set of rules, and you verify for yourself if they are also correct for you. If one of the proposed transactions breaks your own rules, then you mark it as invalid, disconnect from and ban the node who sent you the malicious transaction.

Claim your monetary sovereignty

With your full node you define, verify, and enforce the rules of your sound money Bitcoin.

Notice two core components of this view:

1. A version of the chain that does not accept the rules that you consider fundamental and non-negotiable is *by definition* not bitcoin (or not ethereum or whatever other chain), not matter how many other people accept that chain.
2. What matters is that *you* remain on a chain that has rules that *you* consider acceptable.

However, I believe this "individualist" view to be very wrong. To see why, let us take a look at the scenario that we are worried about: the vast majority of participants accept some change to protocol rules that you find unacceptable. For example, imagine a future where transaction fees are very low, and to keep the chain secure, almost everyone else agrees to change to a new set of rules that increases issuance. You stubbornly keep running a node that continues to enforce the old rules, and you fork off to a different chain than the majority.

From your point of view, you still have your coins in a system that runs on rules that you accept. But so what? Other users will not accept your coins. Exchanges will not accept your coins. Public websites may show the price of the new coin as being some high value, but they're referring to the coins on the majority chain; *your* coins are valueless. Cryptocurrencies and blockchains are fundamentally social constructs; without other people believing in them, they mean nothing.
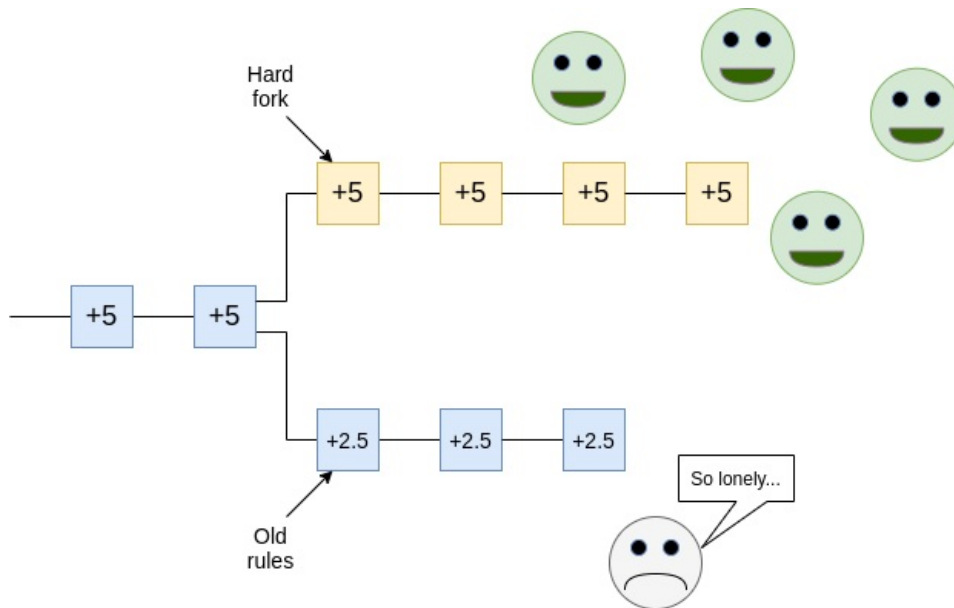
So what is the alternative view? The core idea is to look at blockchains as **engineering security through coordination problems**.

Normally, coordination problems in the world are a bad thing: while it would be better for most people if the English language got rid of its highly complex and irregular spelling system and made a phonetic one, or if the United States switched to metric, or if we could immediately drop all prices and wages by ten percent in the event of a recession, in practice this requires everyone to agree on the switch at the same time, and this is often very very hard.

With blockchain applications, however, *we are using coordination problems to our advantage*. We are using the friction that coordination problems create as a bulwark against malfeasance by centralized actors. We can build systems that have property X, and we can guarantee that they will preserve property X because changing the rules from X to not-X would require a whole bunch of people to agree to update their software at the same time. Even if there is an actor that could force the change, doing so would be hard - much much harder than it would be if it were instead the responsibility of *users* to actively coordinate dissent to resist a change.

Note one particular consequence of this view: it's emphatically *not* the case that the purpose of your full node is just to protect *you*, and in the case of a contentious hard fork, people with full nodes are safe and people without full nodes are vulnerable. Rather, the perspective here is much more one of **herd immunity**: the more people are validating, the more safe *everyone* is, and even if only some portion of people are validating, everyone gets a high level of protection as a result.

Looking deeper into validation

We now get to the next topic, and one that is very relevant to topics such as light clients and sharding: what are we actually accomplishing by validating? To understand this, let us go back to an earlier point. If an attack happens, I would argue that we have the following preference order over how the attack goes:

default to failure > default to chaos > default to victory

The ">" here of course means "better than". The best is if an attack outright fails; the second best is if an attack leads to confusion, with everyone disagreeing on what the correct chain is, and the worst is if an attack succeeds. Why is chaos so much better than victory? This is a matter of incentives: chaos raises costs for the attacker, and denies them the certainty that they will even win, discouraging attacks from being attempted in the first place. A default-to-chaos environment means

that an attacker needs to win *both* the blockchain war of making a 51% attack *and* the "social war" of convincing the community to follow along. This is much more difficult, and much less attractive, than just launching a 51% attack and claiming victory right there.

The goal of validation is then to move away from default to victory to (ideally) default to failure or (less ideally) default to chaos. If you have a fully validating node, and an attacker tries to push through a chain with different rules, then the attack fails. If some people have a fully validating node but many others don't, the attack leads to chaos. But now we can think: are there other ways of achieving the same effect?

Light clients and fraud proofs

One natural advancement in this regard is **light clients with fraud proofs**. Most blockchain light clients that exist today work by simply validating that the majority of miners support a particular block, and not bothering to check if the other protocol rules are being enforced. The client runs on the trust assumption that the majority of miners is honest. If a contentious fork happens, the client follows the majority chain by default, and it's up to users to take an active step if they want to follow the minority chain with the old rules; hence, today's light clients under attack default to victory. But with fraud proof technology, the situation starts to look very different.

A fraud proof in its simplest form works as follows. Typically, a single block in a blockchain only touches a small portion of the blockchain "state" (account balances, smart contract code....). If a fully verifying node processes a block and finds that it is invalid, they can generate a package (the fraud proof) containing the block along with just enough data from the blockchain state to process the block. They broadcast this package to light clients. Light clients can then take the package and use that data to verify the block themselves, even if they have no other data from the chain.
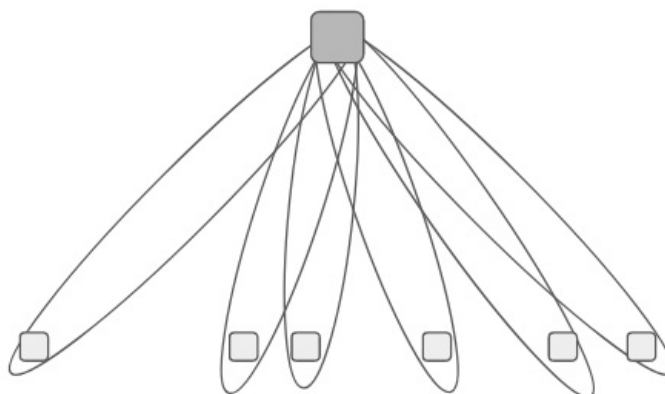
A single block in a blockchain touches only a few accounts. A fraud proof would contain the data in those accounts along with Merkle proofs proving that that data is correct.

This technique is also sometimes known as *stateless validation*: instead of keeping a full database of the blockchain state, clients can keep only the block headers, and they can verify any block in real time by asking other nodes for a Merkle proof for any desired state entries that block validation is accessing.

The power of this technique is that **light clients can verify individual blocks only if they hear an alarm** (and alarms are verifiable, so if a light client hears a false alarm, they can just stop listening to alarms from that node). Hence, under normal circumstances, the light client is still light, checking only which blocks are supported by the majority of miners/validators. But under those exceptional circumstances where the majority chain contains a block that the light client would not accept, **as long as there is at least one honest node verifying the fraudulent block, that node will see that it is invalid, broadcast a fraud proof, and thereby cause the rest of the network to reject it**.

Sharding

Sharding is a natural extension of this: in a sharded system, there are too many transactions in the system for most people to be verifying directly all the time, but if the system is well designed then any individual invalid block can be detected and its invalidity proven with a fraud proof, and that proof can be broadcasted across the entire network. A sharded network can be summarized as

everyone being a light client. And as long as each shard has some minimum threshold number of participants, the network has herd immunity.

In addition, the fact that in a sharded system block *production* (and not just block *verification*) is highly accessible and can be done even on consumer laptops is also very important. The lack of dependence on high-performance hardware at the core of the network ensures that there is a low bar on dissenting minority chains being viable, making it even harder for a majority-driven protocol change to "win by default" and bully everyone else into submission.

This is what auditability usually means in the real world: not that everyone is verifying everything all the time, but that (i) there are enough eyes on each specific piece that if there is an error it will get detected, and (ii) when an error is detected that fact that be made clear and visible to all.

That said, in the long run blockchains can certainly improve on this. One particular source of improvements is ZK-SNARKs (or "validity proofs"): efficiently verifiably cryptographic proofs that allow block producers to prove to clients that blocks satisfy some arbitrarily complex validity conditions. Validity proofs are stronger than fraud proofs because they do not depend on an interactive game to catch fraud. Another important technology is data availability checks, which can protect against blocks whose data is not fully published. Data availability checks do rely on a very conservative assumption that there exists at least some small number of honest nodes somewhere in the network continues to apply, though the good news is that this minimum honesty threshold is low, and does not grow even if there is a very large number of attackers.

Timing and 51% attacks

Now, let us get to the most powerful consequence of the "default to chaos" mindset: 51% attacks themselves. The current norm in many communities is that if a 51% attack wins, then that 51% attack is necessarily the valid chain. This norm is often stuck to quite strictly; and a recent 51% attack on Ethereum Classic illustrated this quite well. The attacker reverted more than 3000 blocks (stealing 807,260 ETC in a double-spend in the process), which pushed the chain farther back in history than one of the two ETC clients (OpenEthereum) was technically able to revert; as a result, Geth nodes went with the attacker's chain but OpenEthereum nodes stuck with the original chain.

We can say that the attack did in fact default to chaos, though this was an accident and not a deliberate design decision of the ETC community. Unfortunately, the community then elected to accept the (longer) attack chain as canonical, a move described by the eth_classic twitter as "following Proof of Work as intended". Hence, *the community norms actively helped the attacker win*.

But we could instead agree on a definition of the canonical chain that works differently: particularly, imagine a rule that once a client has accepted a block as part of the canonical chain, and that block has more than 100 descendants, the client will from then on never accept a chain that does not include that block. Alternatively, in a finality-bearing proof of stake setup (which eg. ethereum 2.0 is), imagine a rule that once a block is finalized it can never be reverted.

5 block revert limit only for illustration purposes; in reality the limit could be something longer like 100-1000 blocks.

To be clear, this introduces a significant change to how canonicalness is determined: instead of clients just looking at the data they receive by itself, clients also look at *when* that data was received. This introduces the possibility that, because of network latency, clients disagree: what if, because of a massive attack, two conflicting blocks A and B finalize at the same time, and some clients see A first and some see B first? But I would argue that this is good: it means that **instead of defaulting to victory, even 51% attacks that just try to revert transactions default to chaos**, and out-of-band emergency response can choose which of the two blocks the chain continues with. If the protocol is well-designed, forcing an escalation to out-of-band emergency response should be very expensive: in proof of stake, such a thing would require 1/3 of validators sacrificing their deposits and getting slashed.

Potentially, we could expand this approach. We could try to [make 51% attacks that censor transactions](#) default to chaos too. Research on **timeliness detectors** pushes things further in the direction of attacks of all types defaulting to failure, though a little chaos remains because timeliness detectors cannot help nodes that are not well-connected and online.

For a blockchain community that values immutability, implementing revert limits of this kind are arguably the superior path to take. It is difficult to honestly claim that a blockchain is immutable when no matter how long a transaction has been accepted in a chain, there is always the possibility that some unexpected activity by powerful actors can come along and revert it. Of course, I would claim that even BTC and ETC do *already* have a revert limit at the extremes; if there was an attack that reverted weeks of activity, the community would likely adopt a user-activated soft fork to reject the attackers' chain. But more definitively agreeing on and formalizing this seems like a step forward.

Conclusion

There are a few "morals of the story" here. First, if we accept the legitimacy of social coordination, and we accept the legitimacy of indirect validation involving "1-of-N" trust models (that is, assuming that there exists one honest person in the network somewhere; NOT the same as assuming that one specific party, eg. Infura, is honest), then we can create blockchains that are much more scalable.

Second, client-side validation is extremely important for all of this to work. A network where only a few people run nodes and everyone else really does trust them is a network that can easily be taken over by special interests. But avoiding such a fate does *not* require going to the opposite extreme and having everyone always validate everything! Systems that allow each individual block to be verified in isolation, so users only validate blocks if someone else raises an alarm, are totally reasonable and serve the same effect. But this requires accepting the "coordination view" of *what validation is for*.

Third, if we allow the definition of canonicalness includes timing, then we open many doors in improving our ability to reject 51% attacks. The easiest property to gain is [weak subjectivity](#): the idea that if clients are required to log on at least once every eg. 3 months, and refuse to revert longer than that, then we can add slashing to proof of stake and make attacks very expensive. But we can go further: we can reject chains that revert finalized blocks and thereby protect immutability, and even protect against censorship. Because the network is unpredictable, relying on timing *does* imply attacks "defaulting to chaos" in some cases, but the benefits are very much worth it.

With all of these ideas in mind, we can avoid the traps of (i) over-centralization, (ii) overly redundant verification leading to inefficiency *and* (iii) misguided norms accidentally making attacks easier, and better work toward building more resilient, performant and secure blockchains.

Gitcoin Grants Round 6 Retrospective

2020 Jul 22
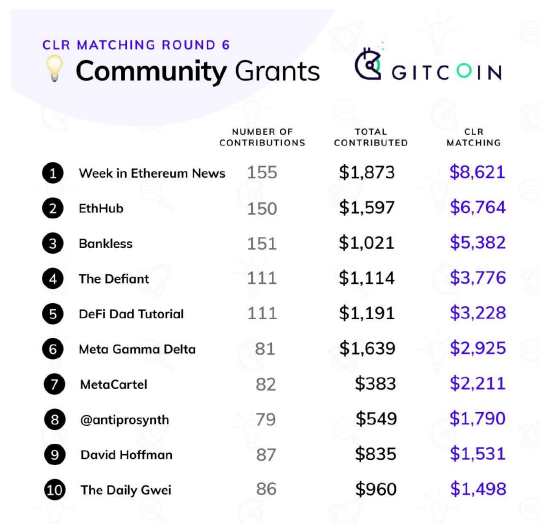
Round 6 of Gitcoin Grants has just finished, with $227,847 in contributions from 1,526 contributors and $175,000 in matched funds distributed across 695 projects. This time around, we had three categories: the two usual categories of "tech" and "community" (the latter renamed from "media" to reflect a desire for a broad emphasis), and the round-6-special category Crypto For Black Lives.

First of all, here are the results, starting with the tech and community sections:

CLR MATCHING ROUND 6
🛠 **Tech** Grants — GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	EIP-1559 Community Fund	412	$29,963	$35,579
2	White Hat Hacking	167	$3,472	$5,728
3	DAppNode	173	$2,820	$5,003
4	Tornado.cash	187	$2,961	$4,965
5	Prysm by Prysmatic Labs	184	$3,127	$4,631
6	Gitcoin Grants Dev Fund	246	$4,518	$4,631
7	1inch.exchange	123	$4,008	$3,281
8	ethers.js	129	$2,857	$2,626
9	Rotki	116	$2,344	$2,508
10	The Commons Stack	107	$5,832	$2,427

CLR MATCHING ROUND 6
💡 **Community** Grants — GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Week in Ethereum News	155	$1,873	$8,621
2	EthHub	150	$1,597	$6,764
3	Bankless	151	$1,021	$5,382
4	The Defiant	111	$1,114	$3,776
5	DeFi Dad Tutorial	111	$1,191	$3,228
6	Meta Gamma Delta	81	$1,639	$2,925
7	MetaCartel	82	$383	$2,211
8	@antiprosynth	79	$549	$1,790
9	David Hoffman	87	$835	$1,531
10	The Daily Gwei	86	$960	$1,498

Stability of income

In the last round, one concern I raised was stability of income. People trying to earn a livelihood off of quadratic funding grants would want to have some guarantee that their income isn't going to completely disappear in the next round just because the hive mind suddenly gets excited about something else.

Round 6 had two mechanisms to try to provide more stability of income:

1. A "shopping cart" interface for giving many contributions, with an explicit "repeat your contributions from the last round" feature
2. A rule that the matching amounts are calculated using not just contributions from this round, but also "carrying over" 1/3 of the contributions from the previous round (ie. if you made a $10

grant in the previous round, the matching formula would pretend you made a $10 grant in the previous round and also a $3.33 grant this round)

1. was clearly successful at one goal: increasing the total number of contributions. But its effect in ensuring stability of income is hard to measure. The effect of (2), on the other hand, is easy to measure, because we have stats for the actual matching amount as well as what the matching amount "would have been" if the 1/3 carry-over rule was not in place.

First from the tech category:

Project	Round 5 match	Round 6 match (with carryover)	Round 6 match (if no carryover)
White Hat Hacking	$15,704	$5,728	$3,676
Arboreum	$9,046	$1,143	$336
1inch.exchange	$7,893	$2,626	$2,429
The Commons Stack	$6,497	$2,426	$1,711
EIP-1559 Community Fund	$0	$35,578	$45,030

Now from the community category:

Project	Round 5 match	Round 6 match (with carryover)	Round 6 match (if no carryover)
Week in Ethereum News	$10,054	$8,621	$7,659
Chris Blec	$5,716	-	-
EthHub	$5,148	$6,764	$6,437
The Defiant	$4,886	$3,776	$3,205
MetaCartel	$3,232	$2,211	$1,797

Clearly, the rule helps reduce volatility, pretty much exactly as expected. That said, one could argue that this result is trivial: you could argue that all that's going on here is something very similar to grabbing part of the revenue from round N (eg. see how the new EIP-1559 Community Fund earned less than it otherwise would have) and moving it into round N+1. Sure, numerically speaking the revenues are more "stable", but individual projects could have just provided this stability to themselves by only spending 2/3 of the pot from each round, and using the remaining third later when some future round is unexpectedly low. Why should the quadratic funding mechanism significantly increase its complexity just to achieve a gain in stability that projects could simply provide for themselves?

My instinct says that it would be best to try the next round with the "repeat last round" feature but *without* the 1/3 carryover, and see what happens. Particularly, note that the numbers seem to show that the media section would have been "stable enough" even without the carryover. The tech section was more volatile, but only because of the sudden entrance of the EIP 1559 community fund; it would be part of the experiment to see just how common that kind of situation is.

About that EIP 1559 Community fund...

The big unexpected winner of this round was the EIP 1559 community fund. EIP 1559 (EIP here, FAQ here, original paper here) is a major fee market reform proposal which far-reaching consequences; it aims to improve the user experience of sending Ethereum transactions, reduce economic

inefficiencies, provide an accurate in-protocol gas price oracle and burn a portion of fee revenue.

Many people in the Ethereum community are very excited about this proposal, though so far there has been fairly little funding toward getting it implemented. This gitcoin grant was a large community effort toward fixing this.

The grant had quite a few very large contributions, including roughly $2,400 each from myself and Eric Conner, early on. Early in the round, one could clearly see the EIP 1559 community grant having an abnormally low ratio of matched funds to contributed funds; it was somewhere around $4k matched to $20k contributed. This was because while the amount contributed was large, it came from relatively few wealthier donors, and so the matching amount was less than it would have been had the same quantity of funds come from more diverse sources - the quadratic funding formula working as intended. However, a social media push advertising the grant then led to a large number of smaller contributors following along, which then quickly raised the match to its currently very high value ($35,578).

Quadratic signaling

Unexpectedly, this grant proved to have a double function. First, it provided $65,473 of much-needed funding to EIP 1559 implementation. Second, it served as a credible community signal of the level of demand for the proposal. The Ethereum community has [long been struggling](#) to find effective ways to determine what "the community" supports, especially in cases of controversy.

Coin votes have been [used in the past](#), and have the advantage that they come with an answer to the key problem of determining who is a "real community member" - the answer is, your membership in the Ethereum community is proportional to how much ETH you have. However, they are plutocratic; in the famous DAO coin vote, a single "yes" voter voted with more ETH than all "no" voters put together (~20% of the total).
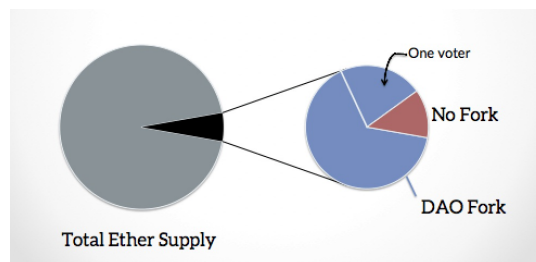
The alternative, looking at github, reddit and twitter comments and votes to measure sentiment (sometimes derided as "proof of social media") is egalitarian, but it is easily exploitable, comes with no skin-in-the-game, and frequently falls under criticisms of "foreign interference" (are those *really* ethereum community members disagreeing with the proposal, or just those dastardly bitcoiners coming in from across the pond to stir up trouble?).

Quadratic funding falls perfectly in the middle: the need to contribute monetary value to vote ensures that the votes of those who *really* care about the project count more than the votes of less-concerned outsiders, and the square-root function ensures that the votes of individual ultra-wealthy "whales" cannot beat out a poorer, but broader, coalition.
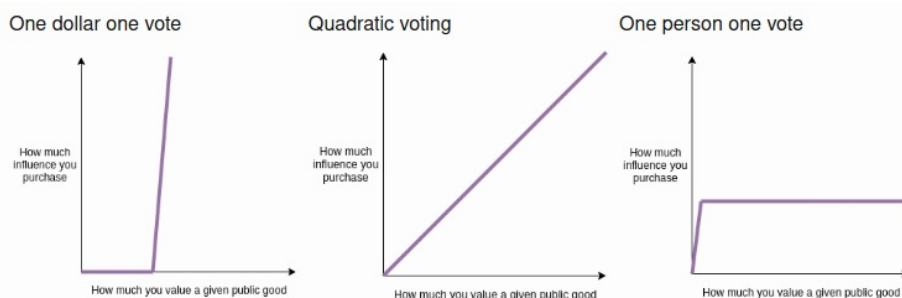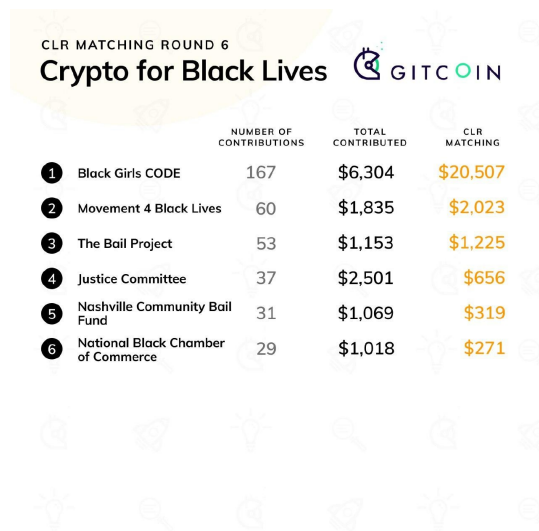
A diagram from my [post on quadratic payments](#) showing how quadratic payments is "in the middle" between the extremes of voting-like systems and money-like systems, and avoids the worst flaws of both.

This raises the question: might it make sense to try to use explicit quadratic *voting* (with the ability to vote "yes" or "no" to a proposal) as an additional signaling tool to determine community sentiment for ethereum protocol proposals?

How well are "guest categories" working?

Since round 5, Gitcoin Grants has had three categories per round: tech, community (called "media" before), and some "guest" category that appears only during that specific round. In round 5 this was COVID relief; in round 6, it's Crypto For Black Lives.

By far the largest recipient was Black Girls CODE, claiming over 80% of the matching pot. My guess for why this happened is simple: Black Girls CODE is an established project that has been participating in the grants for several rounds already, whereas the other projects were new entrants that few people in the Ethereum community knew well. In addition, of course, the Ethereum community "understands" the value of helping people code more than it understands chambers of commerce and bail funds.

This raises the question: is Gitcoin's current approach of having a guest category each round actually working well? The case for "no" is basically this: while the individual causes (empowering black communities, and fighting covid) are certainly admirable, the Ethereum community is by and large not experts at these topics, and we're certainly not experts on *those specific projects* working on those challenges.

If the goal is to try to bring quadratic funding to causes beyond Ethereum, the natural alternative is a separate funding round marketed specifically to those communities; https://downtownstimulus.com/ is a great example of this. If the goal is to get the Ethereum community interested in other causes, then perhaps running more than one round on each cause would work better. For example, "guest categories" could last for three rounds (~6 months), with $8,333 matching per round (and there could be two or three guest categories running simultaneously). In any case, it seems like some revision of the model makes sense.

Collusion

Now, the bad news. This round saw an unprecedented amount of attempted collusion and other forms of fraud. Here are a few of the most egregious examples.

Blatant attempted bribery:
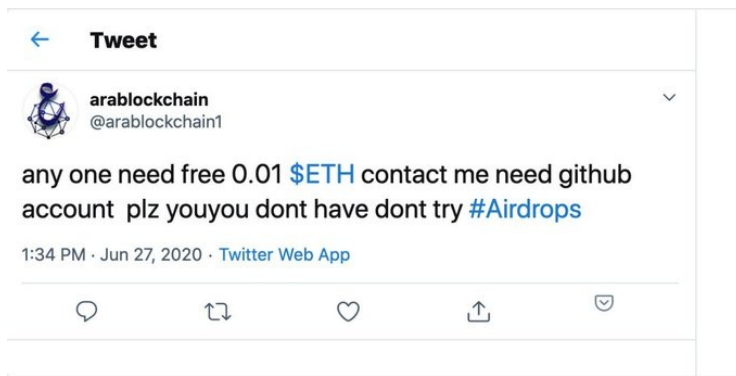
Impersonation:
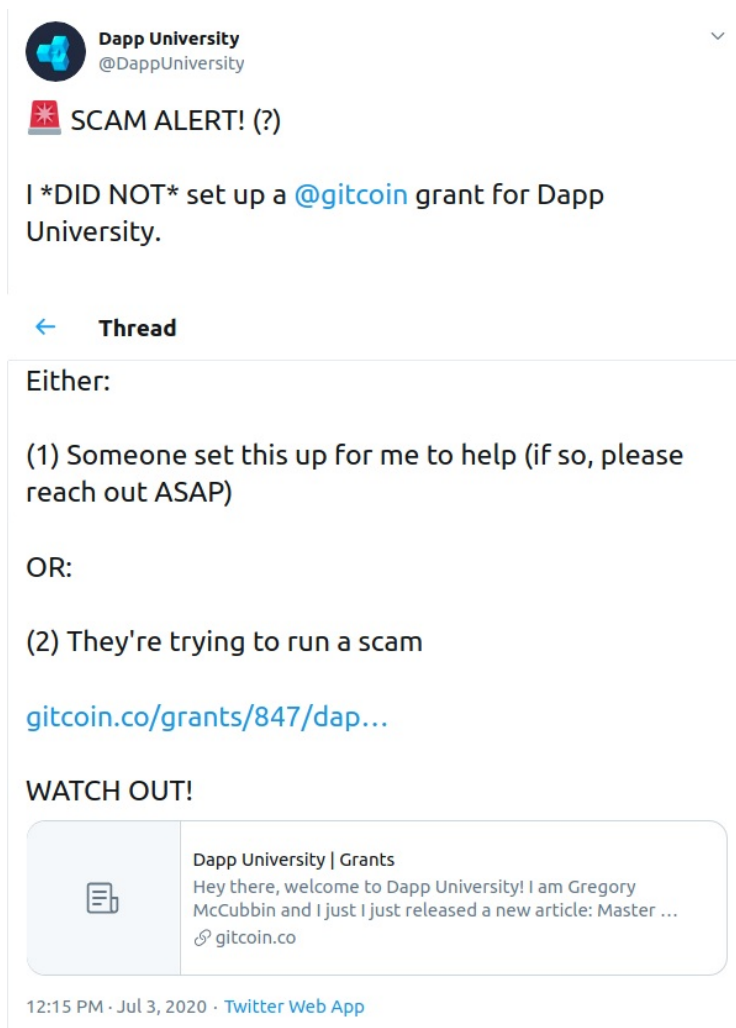
Many contributions with funds clearly coming from a single address:

The big question is: how much fraudulent activity can be prevented in a fully automated/technological way, without requiring detailed analysis of each and every case? If quadratic funding cannot survive such fraud without needing to resort to expensive case-by-case judgement, then regardless of its virtues in an ideal world, in reality it would not be a very good mechanism!

Fortunately, there is a lot that we can do to reduce harmful collusion and fraud that we are not yet doing. Stronger identity systems is one example; in this round, Gitcoin added optional SMS verification, and it seems like the in this round the detected instances of collusion were mostly github-verified accounts and not SMS-verified accounts. In the next round, making some form of extra verification beyond a github account (whether SMS or something more decentralized, eg. BrightID) seems like a good idea. To limit bribery, MACI can help, by making it impossible for a briber to tell who actually voted for any particular project.

Impersonation is not really a quadratic funding-specific challenge; this could be solved with manual verification, or if one wishes for a more decentralized solution one could try using Kleros or some similar system. One could even imagine incentivized reporting: anyone can lay down a deposit and flag a project as fraudulent, triggering an investigation; if the project turns out to be legitimate the deposit is lost but if the project turns out to be fraudulent, the challenger gets half of the funds that were sent to that project.

Conclusion

The best news is the unmentioned news: many of the positive behaviors coming out of the quadratic funding rounds have stabilized. We're seeing valuable projects get funded in the tech and community categories, there has been less social media contention this round than in previous rounds, and people are getting better and better at understanding the mechanism and how to participate in it.

That said, the mechanism is definitely at a scale where we are seeing the kinds of attacks and challenges that we would realistically see in a larger-scale context. There are some challenges that we have not yet worked through (one that I am particularly watching out for is: matched grants going to a project that one part of the community supports and another part of the community thinks is very harmful). That said, we've gotten as far as we have with fewer problems than even I had been anticipating.

I recommend holding steady, focusing on security (and scalability) for the next few rounds, and coming up with ways to increase the size of the matching pots. And I continue to look forward to seeing valuable public goods get funded!

Exploring Fully Homomorphic Encryption

2020 Jul 20

Special thanks to Karl Floersch and Dankrad Feist for review

Fully homomorphic encryption has for a long time been considered one of the holy grails of cryptography. The promise of fully homomorphic encryption (FHE) is powerful: it is a type of encryption that allows a third party to perform computations on encrypted data, and get an encrypted result that they can hand back to whoever has the decryption key for the original data, *without* the third party being able to decrypt the data or the result themselves.
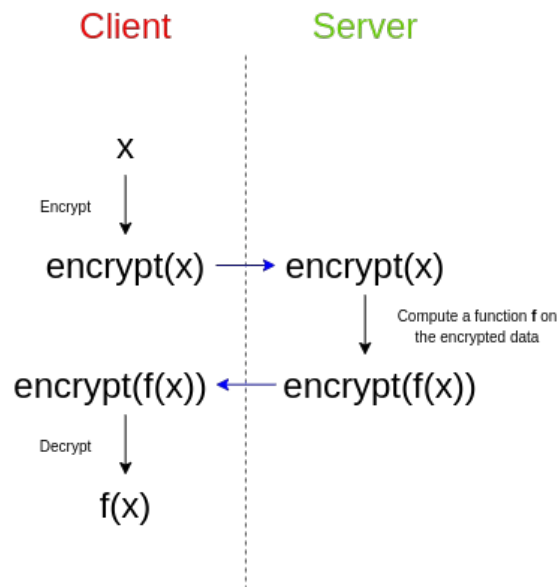
As a simple example, imagine that you have a set of emails, and you want to use a third party spam filter to check whether or not they are spam. The spam filter has a desire for *privacy of their algorithm*: either the spam filter provider wants to keep their source code closed, or the spam filter depends on a very large database that they do not want to reveal publicly as that would make attacking easier, or both. However, you care about the *privacy of your data*, and don't want to upload your unencrypted emails to a third party. So here's how you do it:
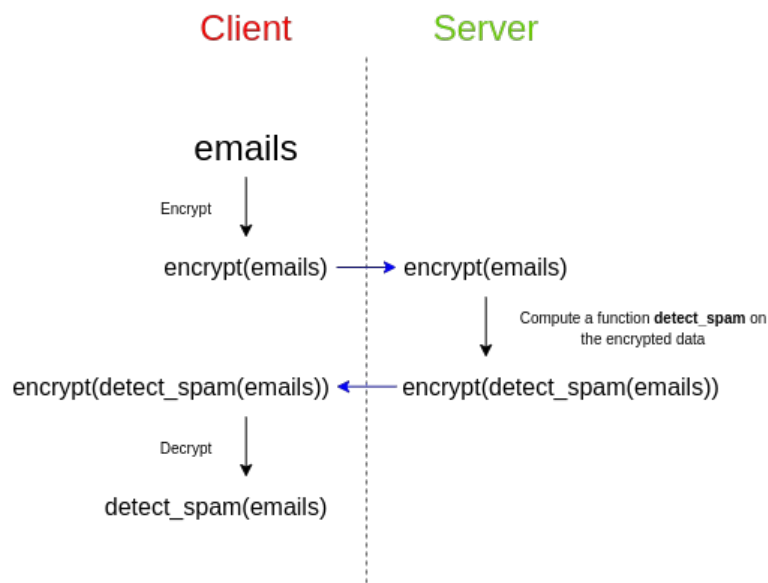
Fully homomorphic encryption has many applications, including in the blockchain space. One key example is that can be used to implement privacy-preserving light clients (the light client hands the server an encrypted index `i`, the server computes and returns `data[0] * (i = 0) + data[1] * (i = 1) + ... + data[n] * (i = n)`, where `data[i]` is the i'th piece of data in a block or state along with its Merkle branch and `(i = k)` is an expression that returns 1 if i = k and otherwise 0; the light client gets the data it needs and the server learns nothing about what the light client asked).

It can also be used for:

- More efficient [stealth address protocols](), and more generally scalability solutions to privacy-preserving protocols that today require each user to personally scan the entire blockchain for incoming transactions
- Privacy-preserving data-sharing marketplaces that let users allow some specific computation to be performed on their data while keeping full control of their data for themselves
- An ingredient in more powerful cryptographic primitives, such as more efficient multi-party computation protocols and perhaps eventually obfuscation

And it turns out that fully homomorphic encryption is, conceptually, not that difficult to understand!

Partially, Somewhat, Fully homomorphic encryption

First, a note on definitions. There are different kinds of homomorphic encryption, some more powerful than others, and they are separated by what kinds of functions one can compute on the encrypted data.

- **Partially homomorphic encryption** allows evaluating only a *very* limited set of operations on encrypted data: either just additions (so given `encrypt(a)` and `encrypt(b)` you can compute `encrypt(a+b)`), or just multiplications (given `encrypt(a)` and `encrypt(b)` you can compute `encrypt(a*b)`).
- **Somewhat homomorphic encryption** allows computing additions as well as a *limited* number of multiplications (alternatively, polynomials up to a limited degree). That is, if you get `encrypt(x1) ... encrypt(xn)` (assuming these are "original" encryptions and not already the result of homomorphic computation), you can compute `encrypt(p(x1 ... xn))`, *as long as* `p(x1 ... xn)` is a polynomial with degree < `D` for some specific degree bound `D` (`D` is usually very low, think 5-15).
- **Fully homomorphic encryption** allows unlimited additions and multiplications. Additions and multiplications let you replicate any binary circuit gates (`AND(x, y) = x*y, OR(x, y) = x+y-x*y, XOR(x, y) = x+y-2*x*y` or just `x+y` if you only care about even vs odd, `NOT(x) = 1-x`...), so this is sufficient to do arbitrary computation on encrypted data.

Partially homomorphic encryption is fairly easy; eg. RSA has a multiplicative homomorphism: $enc(x) = x^e$, $enc(y) = y^e$, so $enc(x) * enc(y) = (xy)^e = enc(xy)$. Elliptic curves can offer similar properties with addition. Allowing *both* addition and multiplication is, it turns out, significantly harder.

A simple somewhat-HE algorithm

Here, we will go through a somewhat-homomorphic encryption algorithm (ie. one that supports a limited number of multiplications) that is surprisingly simple. A more complex version of this category of technique was used by Craig Gentry to create [the first-ever *fully* homomorphic scheme]() in 2009. More recent efforts have switched to using different schemes based on vectors and matrices, but we will still go through this technique first.

We will describe all of these encryption schemes as *secret-key* schemes; that is, the same key is used to encrypt and decrypt. Any secret-key HE scheme can be turned into a public key scheme easily: a "public key" is typically just a set of many encryptions of zero, as well as an encryption of one (and possibly more powers of two). To encrypt a value, generate it by adding together the appropriate subset of the non-zero encryptions, and then adding a random subset of the encryptions of zero to "randomize" the ciphertext and make it infeasible to tell what it represents.

The secret key here is a large prime, p (think of p as having hundreds or even thousands of digits). The scheme can only encrypt 0 or 1, and "addition" becomes XOR, ie. 1 + 1 = 0. To encrypt a value m (which is either 0 or 1), generate a large random value R (this will typically be even larger than p) and a smaller random value r (typically much smaller than p), and output:

$$enc(m) = R * p + r * 2 + m$$

To decrypt a ciphertext ct, compute:

$$dec(ct) = (ct\ mod\ p)\ mod\ 2$$

To add two ciphertexts ct_1 and ct_2, you simply, well, add them: $ct_1 + ct_2$. And to multiply two ciphertexts, you once again... multiply them: $ct_1 * ct_2$. We can prove the homomorphic

property (that the sum of the encryptions is an encryption of the sum, and likewise for products) as follows.

Let:

$$ct_1 = R_1 * p + r_1 * 2 + m_1$$ $$ct_2 = R_2 * p + r_2 * 2 + m_2$$

We add:

$$ct_1 + ct_2 = R_1 * p + R_2 * p + r_1 * 2 + r_2 * 2 + m_1 + m_2$$

Which can be rewritten as:

$$(R_1 + R_2) * p + (r_1 + r_2) * 2 + (m_1 + m_2)$$

Which is of the exact same "form" as a ciphertext of $(m_1 + m_2)$. If you decrypt it, the first $(mod\ p)$ removes the first term, the second $(mod\ 2)$ removes the second term, and what's left is $(m_1 + m_2)$ (remember that if $(m_1 = 1)$ and $(m_2 = 1)$ then the 2 will get absorbed into the second term and you'll be left with zero). And so, voila, we have additive homomorphism!

Now let's check multiplication:

$$ct_1 * ct_2 = (R_1 * p + r_1 * 2 + m_1) * (R_2 * p + r_2 * 2 + m_2)$$

Or:

$$(R_1 * R_2 * p + r_1 * 2 + m_1 + r_2 * 2 + m_2) * p +$$ $$(r_1 * r_2 * 2 + r_1 * m_2 + r_2 * m_1) * 2 +$$ $$(m_1 * m_2)$$

This was simply a matter of expanding the product above, and grouping together all the terms that contain (p), then all the remaining terms that contain (2), and finally the remaining term which is the product of the messages. If you decrypt, then once again the $(mod\ p)$ removes the first group, the $(mod\ 2)$ removes the second group, and only $(m_1 * m_2)$ is left.

But there are two problems here: first, the size of the ciphertext itself grows (the length roughly doubles when you multiply), and second, the "noise" (also often called "error") in the smaller $(* 2)$ term also gets quadratically bigger. Adding this error into the ciphertexts was necessary because the security of this scheme is based on the approximate GCD problem:

Approximate GCD problem

In the **approximate GCD problem** you are given n "near" multiples $\{m_1, m_2, ..., m_n\}$ of some unknown positive integer p, i.e.

$$m_1 = p \cdot q_1 + r_1, \quad 0 \le r_1 < p,$$
$$m_2 = p \cdot q_2 + r_2, \quad 0 \le r_2 < p,$$
$$...,$$
$$m_n = p \cdot q_n + r_n, \quad 0 \le r_n < p,$$

and you need to find out p, without the knowledge of any q_i and r_i.

Had we instead used the "exact GCD problem", breaking the system would be easy: if you just had a set of expressions of the form $(p * R_1 + m_1)$, $(p * R_2 + m_2)$..., then you could use the Euclidean algorithm to efficiently compute the greatest common divisor (p). But if the ciphertexts are only *approximate* multiples of (p) with some "error", then extracting (p) quickly becomes impractical, and so the scheme can be secure.

Unfortunately, the error introduces the inherent limitation that if you multiply the ciphertexts by each other enough times, the error eventually grows big enough that it exceeds (p), and at that point the $(mod\ p)$ and $(mod\ 2)$ steps "interfere" with each other, making the data unextractable. This will be an inherent tradeoff in all of these homomorphic encryption schemes: extracting information from *approximate* equations "with errors" is much harder than extracting information from exact equations, but any error you add quickly increases as you do computations on encrypted data, bounding the amount of computation that you can do before the error becomes overwhelming. And **this is why these schemes are only "somewhat" homomorphic**.

Bootstrapping

There are two classes of solution to this problem. First, in many somewhat homomorphic encryption schemes, there are clever tricks to make multiplication only increase the error by a constant factor (eg.

1000x) instead of squaring it. Increasing the error by 1000x still sounds by a lot, but keep in mind that if p (or its equivalent in other schemes) is a 300-digit number, that means that you can multiply numbers by each other 100 times, which is enough to compute a very wide class of computations. Second, there is Craig Gentry's technique of "bootstrapping".

Suppose that you have a ciphertext ct that is an encryption of some m under a key p, that has a lot of error. The idea is that we "refresh" the ciphertext by turning it into a new ciphertext of m under another key p_2, where this process "clears out" the old error (though it will introduce a fixed amount of new error). The trick is quite clever. The holder of p and p_2 provides a "bootstrapping key" that consists of an encryption of *the bits of p* under the key p_2, as well as the public key for p_2. Whoever is doing computations on data encrypted under p would then take the bits of the ciphertext ct, and individually encrypt these bits under p_2. They would then *homomorphically compute the decryption under p* using these ciphertexts, and get out the single bit, which would be m encrypted under p_2.
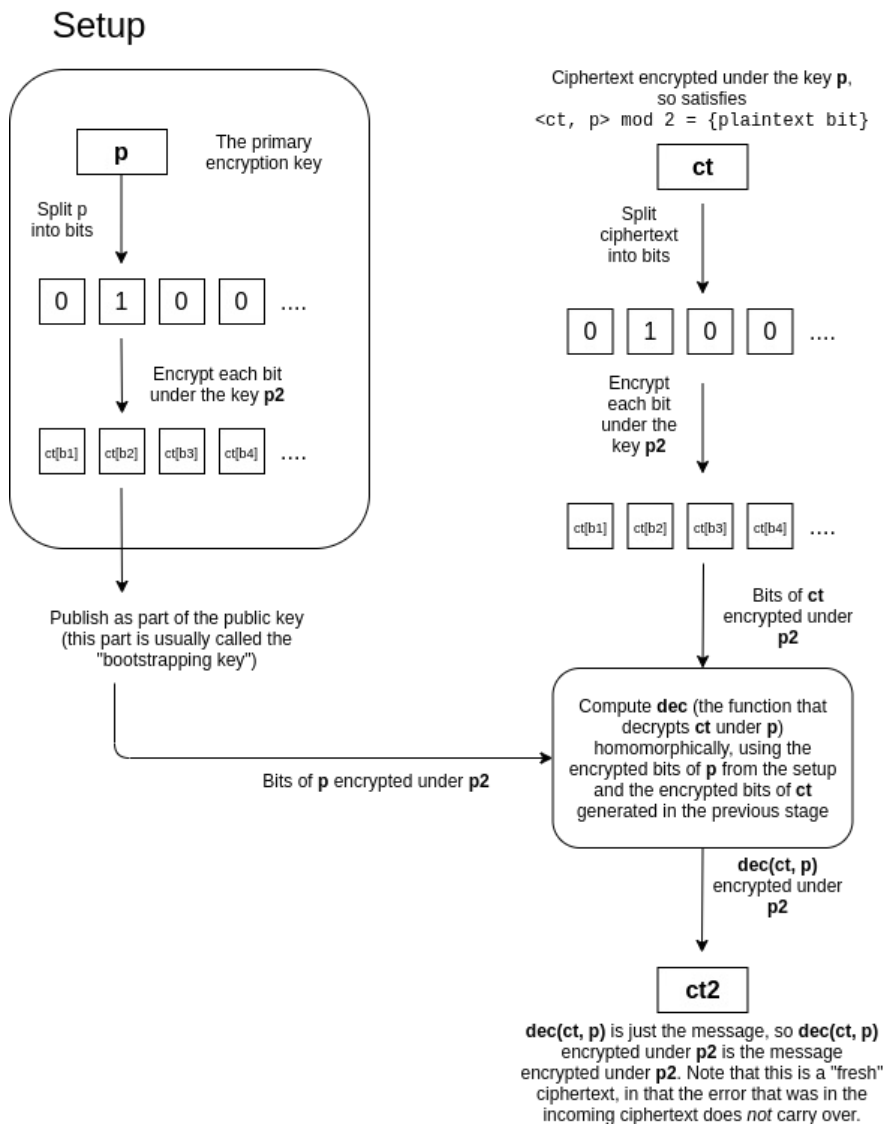
This is difficult to understand, so we can restate it as follows. The decryption procedure $dec(ct, p)$ *is itself a computation*, and so it *can itself be implemented as a circuit* that takes as input the bits of ct and the bits of p, and outputs the decrypted bit $m \in \{0, 1\}$. If someone has a ciphertext ct encrypted under p, a public key for p_2, *and* the bits of p encrypted under p_2, then they can compute $dec(ct, p) = m$ "homomorphically", and get out m encrypted under p_2. Notice that the decryption procedure itself washes away the old error; it just outputs 0 or 1. The decryption procedure is itself a circuit, which contains additions or multiplications, so it will introduce new error, but this new error *does not depend* on the amount of error in the original encryption.

(Note that we can avoid having a distinct new key p_2 (and if you want to bootstrap multiple times, also a p_3, p_4...) by just setting $p_2 = p$. However, this introduces a new assumption, usually called "circular security"; it becomes more difficult to formally prove security if you do this, though many cryptographers think it's fine and circular

security poses no significant risk in practice)

But.... there is a catch. In the scheme as described above (using circular security or not), the error blows up so quickly that even the decryption circuit of the scheme itself is too much for it. That is, the new m encrypted under p_2 would *already* have so much error that it is unreadable. This is because each AND gate doubles the bit-length of the error, so a scheme using a d-bit modulus p can only handle less than $\log(d)$ multiplications (in series), but decryption requires computing $\mod p$ in a circuit made up of these binary logic gates, which requires... more than $\log(d)$ multiplications.

Craig Gentry came up with clever techniques to get around this problem, but they are arguably too complicated to explain; instead, we will skip straight to newer work from 2011 and 2013, that solves this problem in a different way.

Learning with errors

To move further, we will introduce a [different type of somewhat-homomorphic encryption](#) introduced by Brakerski and Vaikuntanathan in 2011, and show how to bootstrap it. Here, we will move away from keys and ciphertexts being *integers*, and instead have keys and ciphertexts be *vectors*. Given a key $k = \{k_1, k_2 .... k_n\}$, to encrypt a message m, construct a vector $c = \{c_1, c_2 ... c_n\}$ such that the inner product (or "[dot product](#)") $<c, k> = c_1k_1 + c_2k_1 + ... + c_nk_n$, modulo some fixed number p, equals $m+2e$ where m is the message (which must be 0 or 1), and e is a small (much smaller than p) "error" term. A "public key" that allows encryption but not decryption can be constructed, as before, by making a set of encryptions of 0; an encryptor can randomly combine a subset of these equations and add 1 if the message they are encrypting is 1. To decrypt a ciphertext c knowing the key k, you would compute $<c, k>$ modulo p, and see if the result is odd or even (this is the same "mod p mod 2" trick we used earlier). Note that here the $\mod p$ is typically a "symmetric" mod, that is, it returns a number between $-\frac{p}{2}$ and $\frac{p}{2}$ (eg. 137 mod 10 = -3, 212 mod 10 = 2); this allows our error to be positive or negative. Additionally, p does not necessarily have to be prime, though it does need to be odd.

Key	3 14 15 92 65
Ciphertext	2 71 82 81 8

The key and the ciphertext are both vectors, in this example of five elements each.

In this example, we set the modulus $p = 103$. The dot product is `3 * 2 + 14 * 71 + 15 * 82 + 92 * 81 + 65 * 8 = 10202`, and $10202 = 99 * 103 + 5$. 5 itself is of course $2 * 2 + 1$, so the message is 1. Note that in practice, the first element of the key is often set to 1; this makes it easier to generate ciphertexts for a particular value (see if you can figure out why).

The security of the scheme is based on an assumption known as "[learning with errors](#)" (LWE) - or, in more jargony but also more understandable terms, the hardness of *solving systems of equations with errors*.

LWE. The LWE problem asks to recover a secret $\mathbf{s} \in \mathbb{Z}_q^n$ given a sequence of 'approximate' random linear equations on $\mathbf{s}$. For instance, the input might be

$$14s_1 + 15s_2 + 5s_3 + 2s_4 \approx 8 \pmod{17}$$
$$13s_1 + 14s_2 + 14s_3 + 6s_4 \approx 16 \pmod{17}$$
$$6s_1 + 10s_2 + 13s_3 + 1s_4 \approx 3 \pmod{17}$$
$$10s_1 + 4s_2 + 12s_3 + 16s_4 \approx 12 \pmod{17}$$
$$9s_1 + 5s_2 + 9s_3 + 6s_4 \approx 9 \pmod{17}$$
$$3s_1 + 6s_2 + 4s_3 + 5s_4 \approx 16 \pmod{17}$$

where each equation is correct up to some small additive error (say, ± 1), and our goal is to recover $\mathbf{s}$. (The answer in this case is $\mathbf{s} = (0, 13, 9, 11)$.)

If not for the error, finding $\mathbf{s}$ would be very easy: after about n equations, we can recover $\mathbf{s}$ in polynomial time using Gaussian elimination. Introducing the error seems to make the problem significantly more difficult. For instance, the Gaussian elimination algorithm takes linear combinations of n equations, thereby amplifying the error to unmanageable levels, leaving essentially no information in the resulting equations.

A ciphertext can itself be viewed as an equation: $k_1 c_1 + ... + k_n c_n \approx 0$, where the key $k_1 ... k_n$ is the unknowns, the ciphertext $c_1 ... c_n$ is the coefficients, and the equality is only approximate because of both the message (0 or 1) and the error ($2e$ for some relatively small e). The LWE assumption ensures that even if you have access to many of these ciphertexts, you cannot recover k.

Note that in some descriptions of LWE, <c, k> can equal any value, but this value must be provided as part of the ciphertext. This is mathematically equivalent to the <c, k> = m+2e formulation, because you can just add this answer to the end of the ciphertext and add -1 to the end of the key, and get two vectors that when multiplied together just give m+2e. We'll use the formulation that requires <c, k> to be near-zero (ie. just m+2e) because it is simpler to work with.

Multiplying ciphertexts

It is easy to verify that the encryption is additive: if $<ct_1, k> = 2e_1 + m_1$ and $<ct_2, k> = 2e_2 + m_2$, then $<ct_1 + ct_2, k> = 2(e_1 + e_2) + m_1 + m_2$ (the addition here is modulo p). What is harder is multiplication: unlike with numbers, there is no natural way to multiply two length-n vectors into another length-n vector. The best that we can do is the outer product: a vector containing the products of each possible pair where the first element comes from the first vector and the second element comes from the second vector. That is, $a \otimes b = a_1 b_1 + a_2 b_1 + ... + a_n b_1 + a_1 b_2 + ... + a_n b_2 + ... + a_n b_n$. We can "multiply ciphertexts" using the convenient mathematical identity $<a \otimes b, c \otimes d> = <a, c> * <b, d>$.

Given two ciphertexts c_1 and c_2, we compute the outer product $c_1 \otimes c_2$. If both c_1 and c_2 were encrypted with k, then $<c_1, k> = 2e_1 + m_1$ and $<c_2, k> = 2e_2 + m_2$. The outer product $c_1 \otimes c_2$ can be viewed as an encryption of $m_1 * m_2$ under $k \otimes k$; we can see this by looking what happens when we try to decrypt with $k \otimes k$:

$$<c_1 \otimes c_2, k \otimes k>$$ $$= <c_1, k> * <c_2, k>$$ $$= (2e_1 + m_1) * (2e_2 + m_2)$$ $$= 2(e_1 m_2 + e_2 m_1 + 2e_1 e_2) + m_1 m_2$$

So this outer-product approach works. But there is, as you may have already noticed, a catch: the size of the ciphertext, and the key, grows quadratically.

Relinearization

We solve this with a **relinearization** procedure. The holder of the private key k provides, as part of the public key, a "relinearization key", which you can think of as "noisy" encryptions of $k \otimes k$ under k. The idea is that we provide these encrypted pieces of $k \otimes k$ to anyone performing the computations, allowing them to compute the equation $<c_1 \otimes c_2, k \otimes k>$ to "decrypt"

the ciphertext, but only in such a way that the output comes back encrypted under k.

It's important to understand what we mean here by "noisy encryptions". Normally, this encryption scheme only allows encrypting $m \in \{0,1\}$, and an "encryption of m" is a vector c such that $<c, k> = m+2e$ for some small error e. Here, we're "encrypting" arbitrary $m \in \{0, 1, 2....p-1\}$. Note that the error means that you can't fully recover m from c; your answer will be off by some multiple of 2. However, it turns out that, for this specific use case, this is fine.

The relinearization key consists of a set of vectors which, when inner-producted (modulo p) with the key k, give values of the form $k_i * k_j * 2^d + 2e$ (mod p), one such vector for every possible triple (i, j, d), where i and j are indices in the key and d is an exponent where $2^d < p$ (note: if the key has length n, there would be $n^2 * \log(p)$ values in the relinearization key; make sure you understand why before continuing).

enc(k$_1$ * k$_1$)	enc(k$_1$ * k$_1$ * 2)	enc(k$_1$ * k$_1$ * 4)	enc(k$_1$ * k$_1$ * 8)
enc(k$_1$ * k2)	enc(k$_1$ * k$_2$ * 2)	enc(k$_1$ * k$_2$ * 4)	enc(k$_1$ * k$_2$ * 8)
enc(k$_2$ * k$_1$)	enc(k$_2$ * k$_1$ * 2)	enc(k$_2$ * k$_1$ * 4)	enc(k$_2$ * k$_1$ * 8)
enc(k$_2$ * k2)	enc(k$_2$ * k$_2$ * 2)	enc(k$_2$ * k$_2$ * 4)	enc(k$_2$ * k$_2$ * 8)

Example assuming p = 15 and k has length 2. Formally, enc(x) here means "outputs x+2e if inner-producted with k".

Now, let us take a step back and look again at our goal. We have a ciphertext which, if decrypted with $k \otimes k$, gives $m_1 * m_2$. We *want* a ciphertext which, if decrypted with k, gives $m_1 * m_2$. We can do this with the relinearization key. Notice that the decryption equation $<ct_1 \otimes ct_2, k \otimes k>$ is just a big sum of terms of the form $(ct_{1_i} * ct_{2_j}) * k_p * k_q$.

And what do we have in our relinearization key? A bunch of elements of the form $2^d * k_p * k_q$, noisy-encrypted under k, for every possible combination of p and q! Having all the powers of two in our relinearization key allows us to generate any $(ct_{1_i} * ct_{2_j}) * k_p * k_q$ by just adding up $\le \log(p)$ powers of two (eg. $13 = 8 + 4 + 1$) together for each (p, q) pair.

For example, if $ct_1 = [1, 2]$ and $ct_2 = [3, 4]$, then $ct_1 \otimes ct_2 = [3, 4, 6, 8]$, and $(enc(<ct_1 \otimes ct_2, k \otimes k>) = enc(3k_1k_1 + 4k_1k_2 + 6k_2k_1 + 8k_2k_2))$ could be computed via:

$$enc(k_1 * k_1) + enc(k_1 * k_1 * 2) + enc(k_1 * k_2 * 4) + $$

$$enc(k_2 * k_1 * 2) + enc(k_2 * k_1 * 4) + enc(k_2 * k_2 * 8) $$

Note that each noisy-encryption in the relinearization key has some even error $2e$, and the equation $<ct_1 \otimes ct_2, k \otimes k>$ itself has some error: if $<ct_1, k> = 2e_1 + m_1$ and $<ct_2 + k> = 2e_2 + m_2$, then $<ct_1 \otimes ct_2, k \otimes k> =$ $<ct_1, k> * <ct_2 + k> =$ $(2(2e_1e_2 + e_1m_2 + e_2m_1) + m_1m_2)$. But this total error is still (relatively) small ($(2e_1e_2 + e_1m_2 + e_2m_1)$ plus $n^2 * \log(p)$ fixed-size errors from the realinearization key), and the error is even, and so the result of this calculation still gives a value which, when inner-producted with k, gives $m_1 * m_2 + 2e'$ for some "combined error" e'.

The broader technique we used here is a common trick in homomorphic encryption: provide pieces of the key encrypted under the key itself (or a different key if you are pedantic about avoiding circular security assumptions), such that someone computing on the data can compute the decryption equation, but only in such a way that the output itself is still encrypted. It was used in bootstrapping above, and it's used here; it's best to make sure you mentally understand what's going on in both cases.

This new ciphertext has considerably more error in it: the $n^2 * \log(p)$ different errors from the portions of the relinearization key that we used, plus the $(2(2e_1e_2 + e_1m_2 + e_2m_1))$ from the original outer-product ciphertext. Hence, the new ciphertext still does have quadratically larger error than the original ciphertexts, and so we still haven't solved the problem that the error blows up too quickly. To solve this, we move on to another trick...

Modulus switching

Here, we need to understand an important algebraic fact. A ciphertext is a vector ct, such that $<ct, k> = m+2e$, where $m \in \{0,1\}$. But we can also look at the ciphertext from a different "perspective": consider $\frac{ct}{2}$ (modulo p). $<\frac{ct}{2}, k> = \frac{m}{2} + e$, where $\frac{m}{2} \in \{0, \frac{p+1}{2}\}$. Note that because (modulo p) $(\frac{p+1}{2})*2 = p+1 = 1$, division by 2 (modulo p) maps 1 to $\frac{p+1}{2}$; this is a very convenient fact for us.

x	Modular division by 2, mod 9	Regular rounded-down division by 2
0	0	0
1	5	0
2	1	1
3	6	1
4	2	2
5	7	2
6	3	3
7	8	3
8	4	4

The scheme in this section uses both modular division (ie. multiplying by the modular multiplicative inverse) and regular "rounded down" integer division; make sure you understand how both work and how they are different from each other.

That is, the operation of dividing by 2 (modulo p) converts small even numbers into small numbers, and it converts 1 into $\frac{p}{2}$ (rounded up). So if we look at $\frac{ct}{2}$ (modulo p) instead of ct, decryption involves computing $<\frac{ct}{2}, k>$ and seeing if it's closer to 0 or $\frac{p}{2}$. This "perspective" is much more robust to certain kinds of errors, where you know the error is small but can't guarantee that it's a multiple of 2.

Now, here is something we can do to a ciphertext.

1. Start: $<ct, k> = \{0\ or\ 1\} + 2e\ (mod\ p)$
2. Divide ct by 2 (modulo p): $<ct', k> = \{0\ or\ \frac{p}{2}\} + e\ (mod\ p)$
3. Multiply ct' by $\frac{q}{p}$ *using "regular rounded-down integer division"*: $<ct'', k> = \{0\ or\ \frac{q}{2}\} + e' + e_2\ (mod\ q)$
4. Multiply ct'' by 2 (modulo q): $<ct''', k> = \{0\ or\ 1\} + 2e' + 2e_2\ (mod\ q)$

Step 3 is the crucial one: it converts a ciphertext under modulus p into a ciphertext under modulus q. The process just involves "scaling down" each element of ct' by multiplying by $\frac{q}{p}$ and rounding down, eg. $floor(56 * \frac{15}{103}) = floor(8.15533..) = 8$.

The idea is this: if $<ct', k> = m*\frac{p}{2} + e\ (mod\ p)$, then we can interpret this as $<ct', k> = p(z + \frac{m}{2}) + e$ for some integer z. Therefore, $<ct' * \frac{q}{p}, k> = q(z + \frac{m}{2}) + e*\frac{p}{q}$. Rounding adds error, but only a little bit (specifically, up to the size of the values in k, and we can make the values in k small without sacrificing security). Therefore, we can say $<ct' * \frac{q}{p}, k> = m*\frac{q}{2} + e' + e_2\ (mod\ q)$, where $e' = e * \frac{q}{p}$, and e_2 is a small error from rounding.

What have we accomplished? We turned a ciphertext with modulus p and error $2e$ into a ciphertext with modulus q and error $2(floor(e*\frac{p}{q}) + e_2)$, where the new error is *smaller* than the original error.

Let's go through the above with an example. Suppose:

- ct is just one value, $[5612]$
- $k = [9]$
- $p = 9999$ and $q = 113$

$<ct, k> = 5612 * 9 = 50508 = 9999 * 5 + 2 * 256 + 1$, so ct represents the bit 1, but the error is fairly large ($e = 256$).

Step 2: $ct' = \frac{ct}{2} = 2806$ (remember this is modular division; if ct were instead 5613, then we would have $\frac{ct}{2} = 7806$). Checking: $<ct', k> = 2806 * 9 = 25254 = 9999 * 2.5 + 256.5$

Step 3: $ct'' = floor(2806 * \frac{113}{9999}) = floor(31.7109...) = 31$. Checking: $<ct'', k> = 279 = 113 * 2.5 - 3.5$

Step 4: $ct''' = 31 * 2 = 62$. Checking: $<ct''', k> = 558 = 113 * 5 - 2 * 4 + 1$

And so the bit 1 is preserved through the transformation. The crazy thing about this procedure is: *none of it requires knowing k*. Now, an astute reader might notice: you reduced the *absolute* size of the error (from 256 to 2), but the *relative* size of the error remained unchanged, and even slightly increased: $\frac{256}{9999} \approx 2.5\%$ but $\frac{4}{113} \approx 3.5\%$. Given that it's the relative error that causes ciphertexts to break, what have we gained here?

The answer comes from what happens to error when you multiply ciphertexts. Suppose that we start with a ciphertext x with error 100, and modulus $p = 10^{16} - 1$. We want to repeatedly square x, to compute $(((x^2)^2)^2)^2 = x^{16}$. First, the "normal way":

Power of x	Error	Modulus
x	100	10^{16}-1
x^2	10^4	10^{16}-1
x^4	10^8	10^{16}-1
x^8	10^{16}	10^{16}-1
x^{16}	10^{32}	10^{16}-1

The error blows up too quickly for the computation to be possible. Now, let's do a modulus reduction after every multiplication. We assume the modulus reduction is imperfect and increases error by a factor of 10, so a 1000x modulo reduction only reduces error from 10000 to 100 (and not to 10):

Power of x	Error	Modulus
x	100	10^{16}-1
x^2	10^4	10^{16}-1
x^2	100	10^{13}-1
x^4	10^4	10^{13}-1
x^4	100	10^{10}-1
x^8	10^4	10^{10}-1
x^8	100	10^7-1
x^{16}	10^4	10^7-1

The key mathematical idea here is that the *factor* by which error increases in a multiplication depends on the absolute size of the error, and not its relative size, and so if we keep doing modulus reductions to keep the error small, each multiplication only increases the error by a constant factor. And so, with a d bit modulus (and hence $\approx 2^d$ room for "error"), we can do $O(d)$ multiplications! This is enough to bootstrap.

Another technique: matrices

Another technique (see [Gentry, Sahai, Waters (2013)](#)) for fully homomorphic encryption involves matrices: instead of representing a ciphertext as ct where $<ct, k> = 2e + m$, a ciphertext is a matrix, where $k * CT = k * m + e$ (k, the key, is still a vector). The idea here is that k is a "secret near-eigenvector" - a secret vector which, if you multiply the matrix by it, returns something very close to either zero or the key itself.

The fact that addition works is easy: if $k * CT\_1 = m\_1 * k + e\_1$ and $k * CT\_2 = m\_2 * k + e\_2$, then $k * (CT\_1 + CT\_2) = (m\_1 + m\_2) * k + (e\_1 + e\_2)$. The fact that multiplication works is also easy:

$(k * CT\_1 * CT\_2)$ $(= (m\_1 * k + e\_1) * CT\_2)$ $(= m\_1 * k * CT\_2 + e\_1 * CT\_2)$ $(= m\_1 * m\_2 * k + m\_1 * e\_2 + e\_1 * CT\_2)$

The first term is the "intended term"; the latter two terms are the "error". That said, notice that here error does blow up quadratically (see the $e\_1 * CT\_2$ term; the size of the error increases by the size of each ciphertext element, and the ciphertext elements also square in size), and you do need some clever tricks for avoiding this. Basically, this involves turning ciphertexts into matrices containing their constituent bits before multiplying, to avoid multiplying by anything higher than 1; if you want to see how this works in detail I recommend looking at my code: https://github.com/vbuterin/research/blob/master/matrix_fhe/matrix_fhe.py#L121

In addition, the code there, and also https://github.com/vbuterin/research/blob/master/tensor_fhe/homomorphic_encryption.py#L186, provides simple examples of useful circuits that you can build out of these binary logical operations; the main example is for adding numbers that are represented as multiple bits, but one can also make circuits for comparison ($<$, $>$, $=$), multiplication, division, and many other operations.

Since 2012-13, when these algorithms were created, there have been many optimizations, but they all work on top of these basic frameworks. Often, polynomials are used instead of integers; this is called ring LWE. The major challenge is still efficiency: an operation involving a single bit involves multiplying entire matrices or performing an entire relinearization computation, a very high overhead. There are tricks that allow you to perform many bit operations in a single ciphertext operation, and this is actively being worked on and improved.

We are quickly getting to the point where many of the applications of homomorphic encryption in privacy-preserving computation are starting to become practical. Additionally, research in the more advanced applications of the lattice-based cryptography used in homomorphic encryption is rapidly progressing. So this is a space where some things can already be done today, but we can hopefully look forward to much more becoming possible over the next decade.

Gitcoin Grants Round 5 Retrospective

2020 Apr 30

Special thanks to Kevin Owocki and Frank Chen for help and review

Round 5 of Gitcoin Grants has just finished, with $250,000 of matching split between tech, media, and the new (non-Ethereum-centric) category of "public health". In general, it seems like the mechanism and the community are settling down into a regular rhythm. People know what it means to contribute, people know what to expect, and the results emerge in a relatively predictable pattern - even if which specific grants get the most funds is not so easy to predict.

CLR MATCHING ROUND 5
🛠 Tech Grants — GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	White Hat Hacking	234	$4,687	$15,704
2	Arboreum	178	$7,038	$9,046
3	1inch.exchange	180	$50,712	$7,893
4	The Commons Stack	145	$3,061	$6,497
5	POAP	128	$2,940	$5,139
6	Tornado.cash	142	$1,178	$3,496
7	DAppNode	107	$1,993	$3,352
8	Rotki	111	$1,578	$3,165
9	Gitcoin Grants Dev	116	$2,688	$3,087
10	Prysm by Prysmatic Labs	119	$1,939	$3,055

CLR MATCHING ROUND 5
💡 Media Grants — GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Week in Ethereum News	157	$2,141	$10,054
2	Chris Blec	143	$1,390	$5,716
3	EthHub	123	$1,566	$5,148
4	The Defiant	116	$3,748	$4,886
5	MetaCartel	99	$610	$3,232
6	Transaction Pending	41	$4,358	$2,547
7	Bankless (Translations)	81	$1,096	$2,167
8	@antiprosynth	68	$1,576	$1,897
9	DeFi Dad Tutorials	73	$465	$1,793
10	Interspace.chat	54	$1,007	$1,755

Stability of income

So let's go straight into the analysis. One important property worth looking at is stability of income across rounds: do projects that do well in round N also tend to do well in round N+1? Stability of income is very important if we want to support an ecosystem of "quadratic freelancers": we want people to feel comfortable relying on their income knowing that it will not completely disappear the next round. On the other hand, it would be harmful if some recipients became completely entrenched, with no opportunity for new projects to come in and compete for the pot, so there is a need for a balance.

On the media side, we do see some balance between stability and dynamism:

Round 4 vs Round 5 income (media)
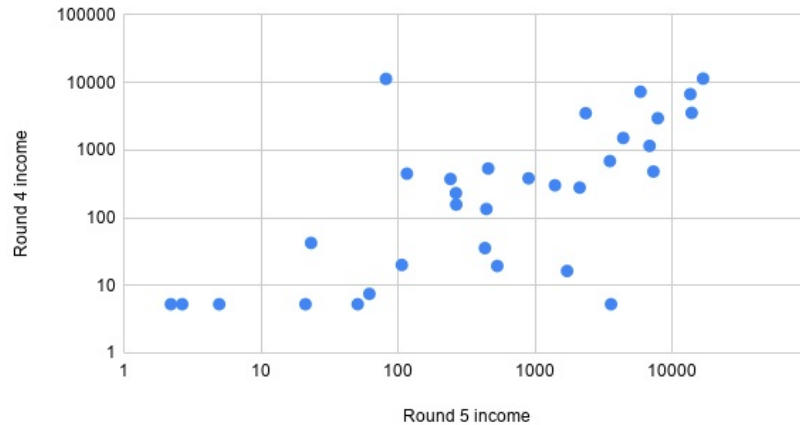
Week in Ethereum had the highest total amount received in both the previous round and the current round. EthHub and Bankless are also near the top in both the current round and the previous round. On the other hand, Antiprosynthesis, the (beloved? notorious? famous?) Twitter info-warrior, has decreased from $13,813 to $5,350, while Chris Blec's YouTube channel has *increased* from $5,851 to $12,803. So some churn, but also some continuity between rounds.

On the tech side, we see much more churn in the winners, with a less clear relationship between income last round and income this round:

Round 4 vs Round 5 income (tech)
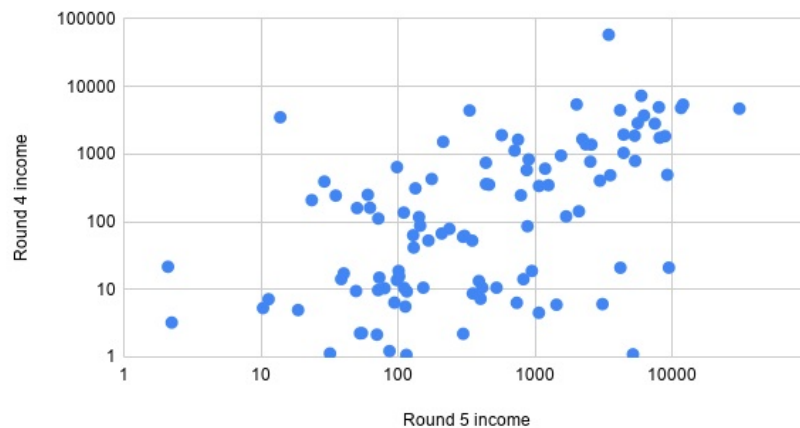
Last round, the winner was Tornado Cash, claiming $30,783; this round, they are down to $8,154. This round, the three roughly-even winners are Samczsun ($4,631 contributions + $15,704 match = $20,335 total), Arboreum ($16,084 contributions + $9,046 match = $25,128 total) and 1inch.exchange ($58,566 contributions + $7,893 match = $66,459 total), in the latter case the bulk coming from one contribution:

In the previous round, those three winners were not even in the top ten, and in some cases not even part of Gitcoin Grants at all.

These numbers show us two things. First, large parts of the Gitcoin community seem to be in the mindset of treating grants not as a question of "how much do you deserve for your last two months of work?", but rather as a one-off reward for years of contributions in the past. This was one of the strongest rebuttals that I received to my criticism of Antiprosynthesis receiving $13,813 in the last round: that the people who

contributed to that award did not see it as two months' salary, but rather as a reward for years of dedication and work for the Ethereum ecosystem. In the next round, contributors were content that the debt was sufficiently repaid, and so they moved on to give a similar gift of appreciation and gratitude to Chris Blec.

That said, not everyone contributes in this way. For example, Prysm got $7,966 last round and $8,033 this round, and Week in Ethereum is consistently well-rewarded ($16,727 previous, $12,195 current), and EthHub saw less stability but still kept half its income ($13,515 previous, $6,705 current) even amid a 20% drop to the matching pool size as some funds were redirected to public health. So there definitely are some contributors that *are* getting almost a reasonable monthly salary from Gitcoin Grants (yes, even these amounts are all serious underpayment, but remember that the pool of funds Gitcoin Grants has to distribute in the first place is quite small, so there's no allocation that would *not* seriously underpay most people; the hope is that in the future we will find ways to make the matching pot grow bigger).

Why didn't more people use recurring contributions?

One feature that was tested this round to try to improve stability was recurring contributions: users could choose to split their contribution among multiple rounds. However, the feature was not used often: out of over 8,000 total contributions, only 120 actually made recurring contributions. I can think of three possible explanations for this:

1. People just don't want to give recurring contributions; they genuinely prefer to freshly rethink who they are supporting every round.
2. People would be willing to give more recurring contributions, but there is some kind of "market failure" stopping them; that is, it's collectively optimal for everyone to give more recurring contributions, but it's not any individual contributor's interest to be the first to do so.
3. There's some UI inconveniences or other "incidental" obstacles preventing recurring contributions.

In a recent call with the Gitcoin team, hypothesis (3) was mentioned frequently. A specific issue was that people were worried about making recurring contributions because they were concerned whether or not the money that they lock up for a recurring contribution would be safe. Improving the payment system and notification workflow may help with this. Another option is to move away from explicit "streaming" and instead simply have the UI provide an option for repeating the last round's contributions and making edits from there.

Hypothesis (1) also should be taken seriously; there's genuine value in preventing ossification and allowing space for new entrants. But I want to zoom in particularly on hypothesis (2), the coordination failure hypothesis.

My explanation of hypothesis (2) starts, interestingly enough, with a defense of (1): why ossification is genuinely a risk. Suppose that there are two projects, A and B, and suppose that they are equal quality. But A already has an established base of contributors; B does not (we'll say for illustration it only has a few existing contributors). Here's how much matching you are contributing by participating in each project:
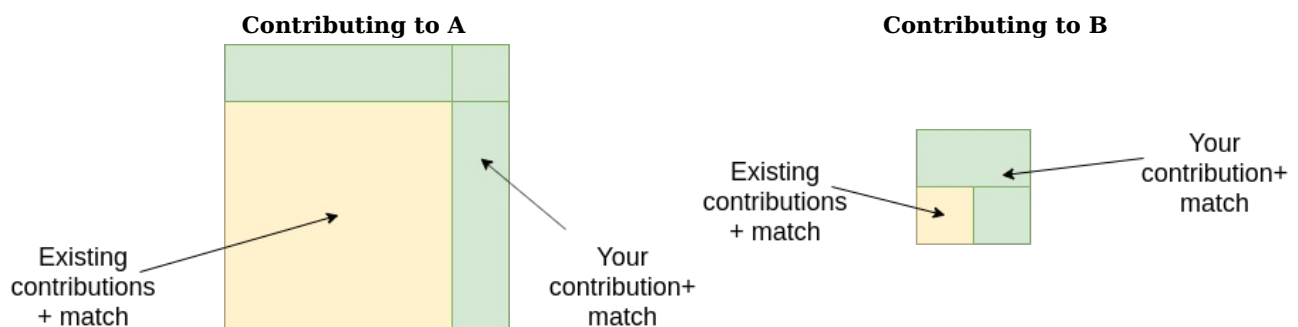
Clearly, you have more impact by supporting A, and so A gets even more contributors and B gets fewer; the rich get richer. Even if project B was *somewhat better*, the greater impact from supporting A could still create a lock-in that reinforces A's position. The current everyone-starts-from-zero-in-each-round mechanism greatly limits this type of entrenchment, because, well, everyone's matching gets reset and starts from zero.

However, a very similar effect also is the cause behind the market failure preventing stable recurring contributions, and the every-round-reset *actually exacerbates it*. Look at the same picture above, except instead of thinking of A and B as *two different projects*, think of them as *the same project in the current round and in the next round*.

We simplify the model as follows. An individual has two choices: contribute $10 in the current round, or contribute $5 in the current round and $5 in the next round. If the matchings in the two rounds were equal, then the latter option would actually be more favorable: because the matching is proportional to the square root of the donation size, the former might give you eg. a $200 match now, but the latter would give you $141 in the current round + $141 in the next round = $282. But if you see a large mass of people contributing in the current round, and you expect much fewer people to contribute in the second round, then the choice is not $200 versus $141 + $141, it might be $200 versus $141 + $5. And so you're better off joining the current

round's frenzy. We can mathematically analyze the equilibrium:

Split between current and next round, vs go all in current round
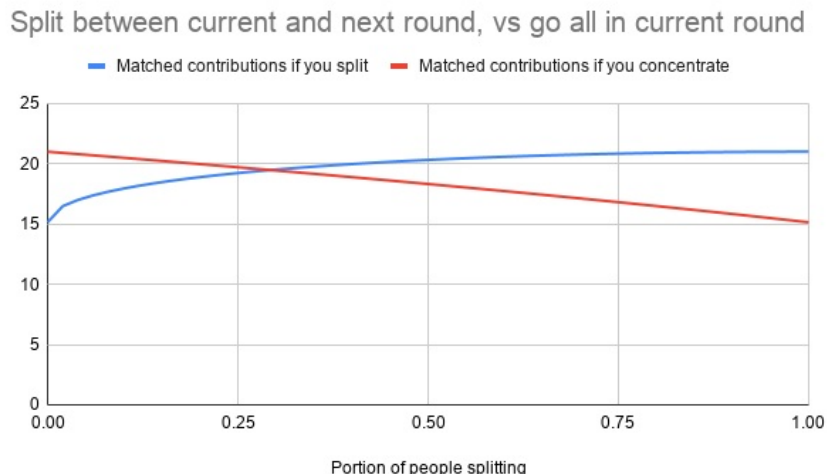
So there is a substantial region within which the bad equilibrium of everyone concentrating is sticky: if more than about 3/4 of contributors are expected to concentrate, it seems in your interest to also concentrate. A mathematically astute reader may note that there is always some intermediate strategy that involves splitting but at a ratio different from 50/50, which you can prove performs better than either full concentrating *or* the even split, but here we get back to hypothesis (3) above: the UI doesn't offer such a complex menu of choices, it just offers the choice of a one-time contribution or a recurring contribution, so people pick one or the other.

How might we fix this? One option is to add a bit of continuity to matching ratios: when computing pairwise matches, match against not just the current round's contributors but, say, 1/3 of the previous round's contributors as well:
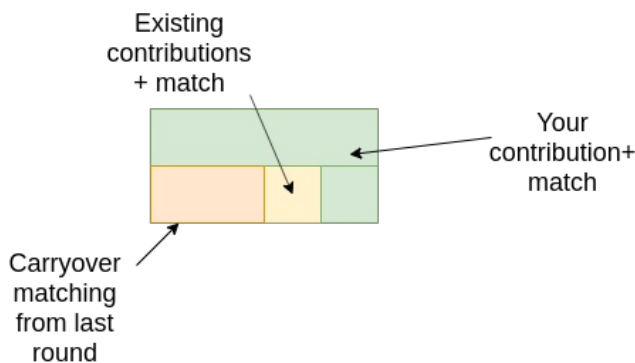
This makes some philosophical sense: the objective of quadratic funding is to subsidize contributions to projects that are detected to be public goods because multiple people have contributed to them, and contributions in the previous round are certainly also evidence of a project's value, so why not reuse those? So here, moving away from everyone-starts-from-zero toward this partial carryover of matching ratios would mitigate the round concentration effect - but, of course, it would exacerbate the risk of entrenchment. Hence, some experimentation and balance may be in order. A broader philosophical question is, is there really a deep inherent tradeoff between risk of entrenchment and stability of income, or is there some way we could get both?

Responses to negative contributions

This round also introduced negative contributions, a feature proposed in my review of the previous round. But as with recurring contributions, very few people made negative contributions, to the point where their impact on the results was negligible. Also, there was active opposition to negative contributions:

Source: honestly I have no idea, someone else sent it to me and they forgot where they found it. Sorry :(

The main source of opposition was basically what I predicted in the previous round. Adding a mechanism that allows people to penalize others, even if deservedly so, can have tricky and easily harmful social consequences. Some people even opposed the negative contribution mechanism to the point where they took care to give positive contributions to everyone who received a negative contribution.

How do we respond? To me it seems clear that, in the long run, *some* mechanism of filtering out bad projects, and ideally compensating for overexcitement into good projects, will have to exist. It doesn't necessarily need to be integrated as a symmetric part of the QF, but there does need to be a filter of some form. And this mechanism, whatever form it will take, invariably opens up the possibility of the same social dynamics. So there is a challenge that will have to be solved no matter how we do it.

One approach would be to hide more information: instead of just hiding *who* made a negative contribution, outright hide the fact that a negative contribution was made. Many opponents of negative contributions explicitly indicated that they would be okay (or at least more okay) with such a model. And indeed (see the next section), this is a direction we will have to go anyway. But it would come at a cost - effectively hiding negative contributions would mean not giving as much real-time feedback into what projects got how much funds.

Stepping up the fight against collusion

This round saw much larger-scale attempts at collusion:

It does seem clear that, at current scales, stronger protections against manipulation are goingto be required. The first thing that can be done is adding a stronger identity verification layer than Github accounts; this is something that the Gitcoin team is already working on. There is definitely a complex tradeoff between security and inclusiveness to be worked through, but it is not especially difficult to implement a first version. And if the identity problem is solved to a reasonable extent, that will likely be enough to prevent collusion at current scales. But in the longer term, we are going to need protection not just against manipulating the system by making many fake accounts, but also against collusion via bribes (explicit and implicit).

MACI is the solution that I proposed (and Barry Whitehat and co are implementing) to solve this problem. Essentially, MACI is a cryptographic construction that allows for contributions to projects to happen on-chain in a privacy-preserving, encrypted form, that allows anyone to cryptographically verify that the mechanism is being implemented *correctly*, but prevents participants from being able to prove to a third party that they made any particular contribution. Unprovability means that if someone tries to bribe others to contribute to their project, the bribe recipients would have no way to prove that they actually contributed to that project, making the bribe unenforceable. Benign "collusion" in the form of friends and family supporting each other would still happen, as people would not easily lie to each other at such small scales, but any broader collusion would be very difficult to maintain.

However, we do need to think through some of the second-order consequences that integrating MACI would introduce. The biggest blessing, and curse, of using MACI is that contributions become hidden. Identities necessarily become hidden, but even the exact timing of contributions would need to be hidden to prevent deanonymization through timing (to prove that *you* contributed, make the total amount jump up between 17:40 and 17:42 today). Instead, for example, totals could be provided and updated once per day. Note that as a corollary negative contributions would be hidden as well; they would only appear if they exceeded all positive contributions for an entire day (and if even that is not desired then the mechanism for when balances are updated could be tweaked to further hide downward changes).

The challenge with hiding contributions is that we lose the "social proof" motivator for contributing: if contributions are unprovable you can't as easily publicly brag about a contribution you made. My best proposal for solving this is for the mechanism to publish one extra number: the *total* amount that a particular participant contributed (counting only projects that have received at least 10 contributors to prevent inflating one's number by self-dealing). Individuals would then have a generic "proof-of-generosity" that they contributed some specific *total* amount, and could publicly state (without proof) what projects it was that they supported. But this is all a significant change to the user experience that will require multiple rounds of experimentation to get right.

Conclusions

All in all, Gitcoin Grants is establishing itself as a significant pillar of the Ethereum ecosystem that more and more projects are relying on for some or all of their support. While it has a relatively low amount of funding at present, and so inevitably underfunds almost everything it touches, we hope that over time we'll continue to see larger sources of funding for the matching pools appear. One option is MEV auctions, another is that new or existing token projects looking to do airdrops could provide the tokens to a matching pool. A third is transaction fees of various applications. With larger amounts of funding, Gitcoin Grants could serve as a more significant funding stream - though to get to that point, further iteration and work on fine-tuning the mechanism will be required.

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	CIC (COVID-19) Kenyan Crisis Aid	85	$2,767	$19,802
2	Open Source Covid Ventilators + Masks	89	$2,472	$19,628
3	Mask + Test Kit Mutual Aid Fund	87	$2,447	$16,718
4	CuraDAO COVID-19 Campaign	75	$1,056	$9,884
5	Giveth & Coz: Giving to COVID-19 Causes	57	$6,027	$6,785
6	African Angels	48	$1,970	$5,947
7	Decentralised supply chain of 3D printed protective gear	49	$700	$4,322
8	Tracy	93	$3,186	$4,058
9	Save the Children	52	$6,963	$3,344
10	Collab19	35	$175	$1,844

Additionally, this round saw Gitcoin Grants' first foray into applications beyond Ethereum with the health section. There is growing interest in quadratic funding from local government bodies and other non-blockchain groups, and it would be very valuable to see quadratic funding more broadly deployed in such contexts. That said, there are unique challenges there too. First, there's issues around onboarding people who do not already have cryptocurrency. Second, the Ethereum community is naturally expert in the needs of the Ethereum community, but neither it nor average people are expert in, eg. medical support for the coronavirus pandemic. We should expect quadratic funding to perform worse when the participants are not experts in the domain they're being asked to contribute to. Will non-blockchain uses of QF focus on domains where there's a clear local community that's expert in its own needs, or will people try larger-scale deployments soon? If we do see larger-scale deployments, how will those turn out? There's still a lot of questions to be answered.

A Quick Garbled Circuits Primer

2020 Mar 21

Special thanks to Dankrad Feist for review

[Garbled circuits](#) are a quite old, and surprisingly simple, cryptographic primitive; they are quite possibly the simplest form of general-purpose "multi-party computation" (MPC) to wrap your head around.

Here is the usual setup for the scheme:

- Suppose that there are two parties, Alice and Bob, who want to compute some function `f(alice_inputs, bob_inputs)`, which takes inputs from both parties. Alice and Bob want to both learn the result of computing `f`, but Alice does not want Bob to learn her inputs, and Bob does not want Alice to learn his inputs. Ideally, they would both learn nothing except for just the output of `f`.
- Alice performs a special procedure ("garbling") to encrypt a circuit (meaning, a set of AND, OR... gates) which evaluates the function `f`. She passes along inputs, also encrypted in a way that's compatible with the encrypted circuit, to Bob.
- Bob uses a technique called "1-of-2 oblivious transfer" to learn the encrypted form of his own inputs, without letting Alice know which inputs he obtained.
- Bob runs the encrypted circuit on the encrypted data and gets the answer, and passes it along to Alice.

Extra cryptographic wrappings can be used to protect the scheme against Alice and Bob sending wrong info and giving each other an incorrect answer; we won't go into those here for simplicity, though it suffices to say "wrap a ZK-SNARK around everything" is one (quite heavy duty and suboptimal!) solution that works fine.

So how does the basic scheme work? Let's start with a circuit:
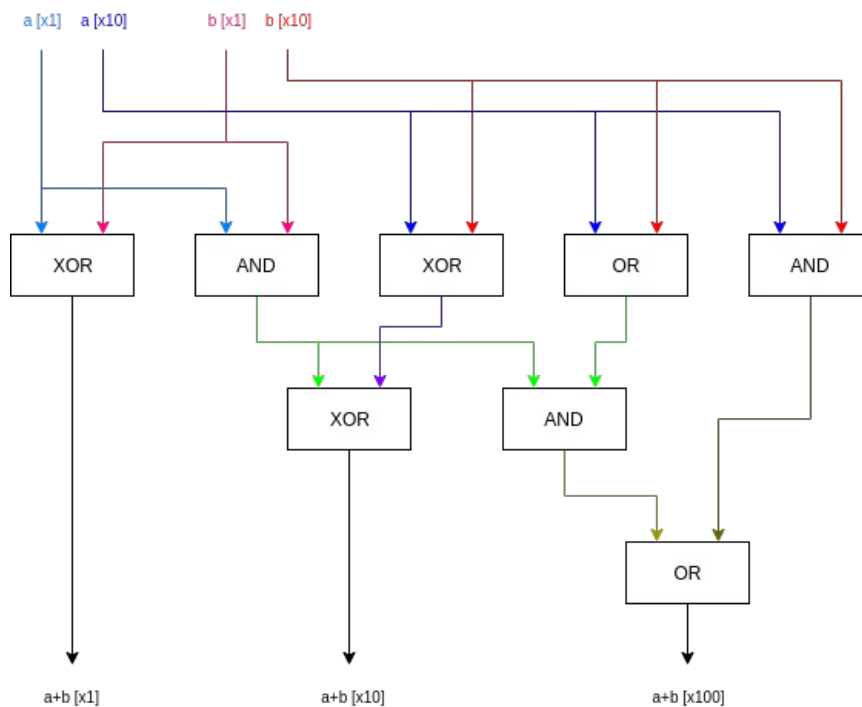
This is one of the simplest examples of a not-completely-trivial circuit that actually does something: it's a two-bit adder. It takes as input two numbers in binary, each with two bits, and outputs the three-bit binary number that is the sum.

Now, let's encrypt the circuit. First, for every input, we randomly generate two "labels" (think: 256-bit numbers): one to represent that input being 0 and the other to represent that input being 1. Then we also do the same for every intermediate wire, not including the output wires. Note that this data is not part of the "garbling" that Alice sends to Bob; so far this is just setup.
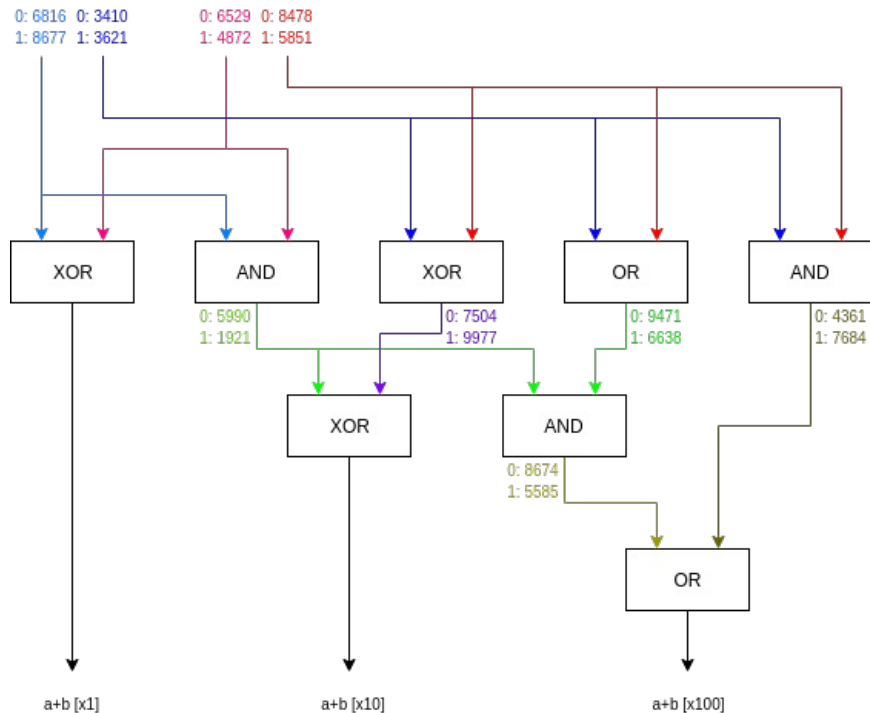
Now, for every gate in the circuit, we do the following. For every combination of inputs, we include in the "garbling" that Alice provides to Bob the label of the output (or if the label of the output is a "final" output, the output directly) encrypted with a key generated by hashing the input labels that lead to that output together. For simplicity, our encryption algorithm can just be enc(out, in1, in2) = out + hash(k, in1, in2) where k is the index of the gate (is it the first gate in the circuit, the second, the third?). If you know the labels of both inputs, and you have the garbling, then you can learn the label of the corresponding output, because you can just compute the corresponding hash and subtract it out.

Here's the garbling of the first XOR gate:

Inputs	Output	Encoding of output
00	0	0 + hash(1, 6816, 6529)
01	1	1 + hash(1, 6816, 4872)
10	1	1 + hash(1, 8677, 6529)
11	0	0 + hash(1, 8677, 4872)

Notice that we are including the (encrypted forms of) 0 and 1 directly, because this XOR gate's outputs are directly final outputs of the program. Now, let's look at the leftmost AND gate:

Inputs	Output	Encoding of output
00	0	5990 + hash(2, 6816, 6529)
01	0	5990 + hash(2, 6816, 4872)
10	0	5990 + hash(2, 8677, 6529)
11	1	1921 + hash(2, 8677, 4872)

Here, the gate's outputs are just used as inputs to other gates, so we use labels instead of bits to hide these intermediate bits from the evaluator.

The "garbling" that Alice would provide to Bob is just everything in the third column for each gate, with the rows of each gate re-ordered (to avoid revealing whether a given row corresponds to a 0 or a 1 in any wire). To help Bob learn which value to decrypt for each gate, we'll use a particular order: for each gate, the first row becomes the row where both input labels are even, in the second row the

second label is odd, in the third row the first label is odd, and in the fourth row both labels are odd (we deliberately chose labels earlier so that each gate would have an even label for one output and an odd label for the other). We garble every other gate in the circuit in the same way.

All in all, Alice sends to Bob four ~256 bit numbers for each gate in the circuit. It turns out that four is far from optimal; see here for some optimizations on how to reduce this to three or even two numbers for an AND gate and zero (!!) for an XOR gate. Note that these optimizations do rely on some changes, eg. using XOR instead of addition and subtraction, though this should be done anyway for security.

When Bob receives the circuit, he asks Alice for the labels corresponding to her input, and he uses a protocol called "1-of-2 oblivious transfer" to ask Alice for the labels corresponding to his own input without revealing to Alice what his input is. He then goes through the gates in the circuit one by one, uncovering the output wires of each intermediate gate.

Suppose Alice's input is the two left wires and she gives (0, 1), and Bob's input is the two right wires and he gives (1, 1). Here's the circuit with labels again:
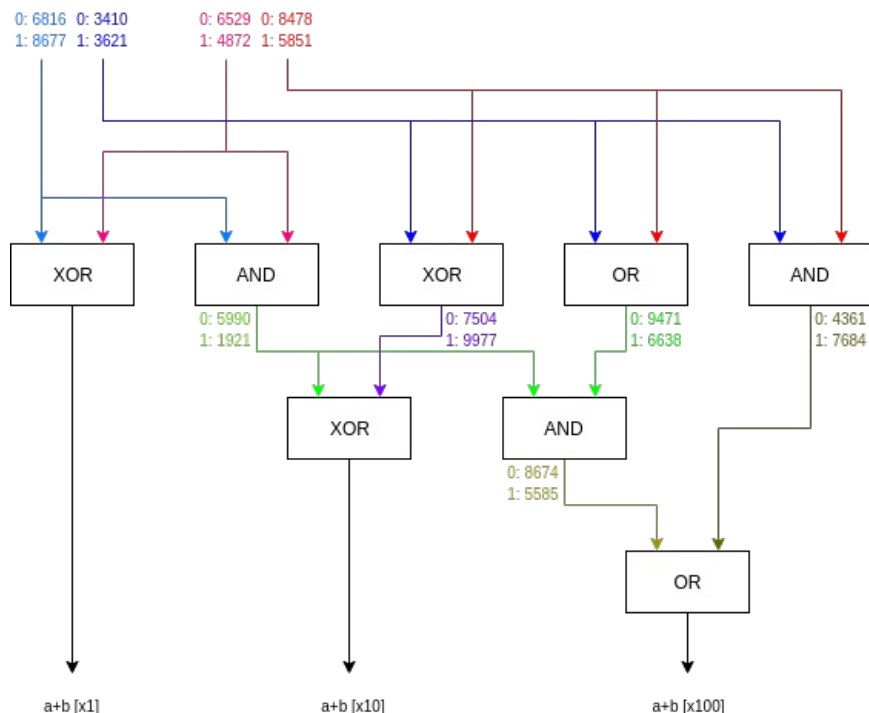
- At the start, Bob knows the labels 6816, 3621, 4872, 5851
- Bob evaluates the first gate. He knows 6816 and 4872, so he can extract the output value corresponding to (1, 6816, 4872) (see the table above) and extracts the first output bit, 1
- Bob evaluates the second gate. He knows 6816 and 4872, so he can extract the output value corresponding to (2, 6816, 4872) (see the table above) and extracts the label 5990
- Bob evaluates the third gate (XOR). He knows 3621 and 5851, and learns 7504
- Bob evaluates the fourth gate (OR). He knows 3621 and 5851, and learns 6638
- Bob evaluates the fifth gate (AND). He knows 3621 and 5851, and learns 7684
- Bob evaluates the sixth gate (XOR). He knows 5990 and 7504, and learns the second output bit, 0
- Bob evaluates the seventh gate (AND). He knows 5990 and 6638, and learns 8674
- Bob evaluates the eighth gate (OR). He knows 8674 and 7684, and learns the third output bit, 1

And so Bob learns the output: 101. And in binary 10 + 11 actually equals 101 (the input and output bits are both given in smallest-to-greatest order in the circuit, which is why Alice's input 10 is represented as (0, 1) in the circuit), so it worked!

Note that addition is a fairly pointless use of garbled circuits, because Bob knowing 101 can just subtract out his own input and get 101 - 11 = 10 (Alice's input), breaking privacy. However, in general garbled circuits can be used for computations that are not reversible, and so don't break privacy in this way (eg. one might imagine a computation where Alice's input and Bob's input are their answers to a personality quiz, and the output is a single bit that determines whether or not the

algorithm thinks they are compatible; that one bit of information won't let Alice or Bob know anything about each other's individual quiz answers).

1 of 2 Oblivious Transfer

Now let us talk more about 1-of-2 oblivious transfer, this technique that Bob used to obtain the labels from Alice corresponding to his own input. The problem is this. Focusing on Bob's first input bit (the algorithm for the second input bit is the same), Alice has a label corresponding to 0 (6529), and a label corresponding to 1 (4872). Bob has his desired input bit: 1. Bob wants to learn the correct label (4872) without letting Alice know that his input bit is 1. The trivial solution (Alice just sends Bob both 6529 and 4872) doesn't work because Alice only wants to give up one of the two input labels; if Bob receives both input labels this could leak data that Alice doesn't want to give up.

Here is a fairly simple protocol using elliptic curves:

1. Alice generates a random elliptic curve point, `H`.
2. Bob generates two points, `P1` and `P2`, with the requirement that `P1 + P2` sums to `H`. Bob chooses either `P1` or `P2` to be `G * k` (ie. a point that he knows the corresponding private key for). Note that the requirement that `P1 + P2 = H` ensures that Bob has no way to generate `P1` and `P2` such that he knows the corresponding private key for. This is because if `P1 = G * k1` and `P2 = G * k2` where Bob knows both `k1` and `k2`, then `H = G * (k1 + k2)`, so that would imply Bob can extract the discrete logarithm (or "corresponding private key") for `H`, which would imply all of elliptic curve cryptography is broken.
3. Alice confirms `P1 + P2 = H`, and encrypts `v1` under `P1` and `v2` under `P2` using some standard public key encryption scheme (eg. El-Gamal). Bob is only able to decrypt one of the two values, because he knows the private key corresponding to at most one of (`P1`, `P2`), but Alice does not know which one.

This solves the problem; Bob learns one of the two wire labels (either 6529 or 4872), depending on what his input bit is, and Alice does not know which label Bob learned.

Applications

Garbled circuits are potentially useful for many more things than just 2-of-2 computation. For example, you can use them to make multi-party computations of arbitrary complexity with an arbitrary number of participants providing inputs, that can run in a constant number of rounds of interaction. Generating a garbled circuit is completely parallelizable; you don't need to finish garbling one gate before you can start garbling gates that depend on it. Hence, you can simply have a large multi-party computation with many participants compute a garbling of all gates of a circuit and publish the labels corresponding to their inputs. The labels themselves are random and so reveal nothing about the inputs, but anyone can then execute the published garbled circuit and learn the output "in the clear". See here for a recent example of an MPC protocol that uses garbling as an ingredient.

Multi-party computation is not the only context where this technique of splitting up a computation into a parallelizable part that operates on secret data followed by a sequential part that can be run in the clear is useful, and garbled circuits are not the only technique for accomplishing this. In general, the literature on randomized encodings includes many more sophisticated techniques. This branch of math is also useful in technologies such as functional encryption and obfuscation.

Review of Gitcoin Quadratic Funding Round 4

2020 Jan 28

Round 4 of Gitcoin Grants quadratic funding has just completed, and here are the results:

CLR MATCHING ROUND 4
Tech Grants GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Tornado.cash	308	$3,648	$27,135
2	DAppNode	185	$2,984	$8,583
3	Sablier	172	$1,401	$8,020
4	MetaGame	173	$1,226	$7,609
5	DeFiZap	177	$2,369	$6,807
6	Gitcoin Sustainability	137	$7,120	$4,840
7	Commons Simulator	118	$1,727	$4,475
8	Prysm	140	$3,601	$4,365
9	Uniswap	124	$1,715	$3,439
10	Enzypt.io	102	$900	$3,275

CLR MATCHING ROUND 4
Media Grants GITCOIN

		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Week in Ethereum News	140	$3,191	$13,536
2	@antiprosynth	134	$2,420	$11,393
3	EthHub	139	$2,150	$11,365
4	Bankless (Scholarships)	104	$2,093	$5,733
5	David Hoffman	101	$1,399	$5,414
6	Wizards of DApps	86	$2,088	$5,258
7	Cryptorado	75	$1,475	$4,682
8	Ethereum Magicians	69	$633	$3,745
9	Zero Knowledge Podcast	60	$503	$2,998
10	DeFi by Chris Blec	76	$2,975	$2,876

The main distinction between round 3 and round 4 was that while round 3 had only one category, with mostly tech projects and a few outliers such as EthHub, in round 4 there were two separate categories, one with a $125,000 matching pool for tech projects, and the other with a $75,000 matching pool for "media" projects. Media could include documentation, translation, community activities, news reporting, theoretically pretty much anything in that category. And while the tech section went about largely without incident, in the new media section **the results proved to be much more interesting than I could have possibly imagined, shedding a new light on deep questions in institutional design and political science**.

Tech: quadratic funding worked great as usual

In the tech section, the main changes that we see compared to round 3 are (i) the rise of [Tornado Cash] and (ii) the decline in importance of eth2 clients and the rise of "utility applications" of various forms. Tornado Cash is a trustless smart contract-based Ethereum mixer. It became popular quickly in recent months, as the Ethereum community was swept by worries about the blockchain's current [low levels of privacy] and wanted solutions. Tornado Cash amassed an incredible $31,200. If they continue receiving such an amount every two months then this would allow them to pay two people $7,800 per month each - meaning that the hoped-for milestone of seeing the first "quadratic freelancer" may have already been reached! The other major winners included tools like [Dappnode], a software package to help people run nodes, [Sablier], a payment streaming service, and [DeFiZap], which makes DeFi services easy to use. The [Gitcoin Sustainability Fund] got over $13,000, conclusively resolving my complaint from [last round] that they were under-supported. All in all, valuable grants for valuable projects that provide services that the community genuinely needs.

We can see one major shift this round compared to the previous rounds. Whereas in previous rounds, the grants went largely to projects like eth2 clients that were already well-supported, this time the largest grants shifted toward having a different focus from the grants given by the Ethereum Foundation. The EF has not given grants to tornado.cash, and generally limits its grants to application-specific tools, Uniswap being a notable exception. The Gitcoin Grants quadratic fund, on the other hand, is supporting DeFiZap, Sablier, and many other tools that are valuable to the community. This is arguably a positive development, as it allows Gitcoin Grants and the Ethereum Foundation to complement each other rather than focusing on the same things.

The one proposed change to the quadratic funding implementation for tech that I would favor is a user interface change,

that makes it easier for users to commit funds for multiple rounds. This would increase the stability of contributions, thereby increasing the stability of projects' income - very important if we want "quadratic freelancer" to actually be a viable job category!

Media: The First Quadratic Twitter Freelancer

Now, we get to the new media section. In the first few days of the round, the leading recipient of the grants was "@antiprosynth Twitter account activity": an Ethereum community member who is [very active on twitter](#) promoting Ethereum and refuting misinformation from Bitcoin maximalists, asking for help from the Gitcoin QF crowd to.... fund his tweeting activities. At its peak, the projected matching going to @antiprosynth exceeded $20,000. This naturally proved to be controversial, with many criticizing this move and questioning whether or not a Twitter account is a legitimate public good:

On the surface, it does indeed seem like someone getting paid $20,000 for operating a Twitter account is ridiculous. But it's worth digging in and questioning exactly *what*, if anything, is actually wrong with this outcome. After all, maybe this is what effective marketing in 2020 actually looks like, and it's our expectations that need to adapt.

There are two main objections that I heard, and both lead to interesting criticisms of quadratic funding in its current implementation. First, there was criticism of **overpayment**. Twittering is a fairly "trivial" activity; it does not require *that* much work, lots of people do it for free, and it doesn't provide nearly as much long-term value as something more substantive like [EthHub](#) or the [Zero Knowledge Podcast](#). Hence, it feels wrong to pay a full-time salary for it.
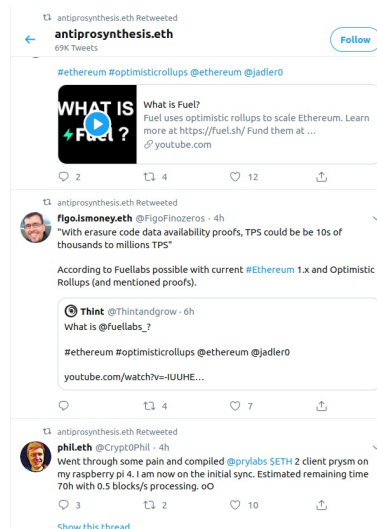
Examples of @antiprosynth's recent tweets

If we accept the metaphor of quadratic funding as being like a [market for public goods](#), then one could simply extend the metaphor, and reply to the concern with the usual free-market argument. People voluntarily paid their own money to support @antiprosynth's twitter activity, and that itself signals that it's valuable. Why should we trust you with your mere words and protestations over a costly signal of real money on the table from dozens of people?

The most plausible answer is actually quite similar to one that you often hear in discussions about financial markets: markets can give skewed results when you can express an opinion *in favor of* something but cannot express an opinion *against* it. When short selling is not possible, financial markets [are often much more inefficient](#), because instead of reflecting the *average* opinion on an asset's true value, a market may instead reflect the inflated expectations of an asset's few rabid supporters. In this version of quadratic funding, **there too is an asymmetry, as you can donate in support of a project but you cannot donate to oppose it**. Might this be the root of the problem?

One can go further and ask, why might overpayment happen to this particular project, and not others? I have heard a common answer: twitter accounts *already have a high exposure*. A client development team like [Nethermind](#) does not gain much publicity through their work directly, so they need to separately market themselves, whereas a twitter account's "work" is self-marketing by its very nature. Furthermore, the most prominent twitterers get quadratically more matching out of their exposure, amplifying their outsized advantage further - a problem I alluded to in my [review of round 3](#).

Interestingly, in the case of vanilla quadratic voting there [was an argument](#) made by Glen Weyl for why economies-of-scale effects of traditional voting, such as [Duverger's law](#), don't apply to quadratic voting: a project becoming more prominent increases the incentive to give it both positive and negative votes, so on net the effects cancel out. But notice once again, that **this argument relies on negative votes being a possibility**.

Good for the tribe, but is it good for the world?

The particular story of @antiprosynth had what is in my opinion a happy ending: over the next ten days, more contributions came in to other candidates, and @antiprosynth's match reduced to $11,316, still a respectably high amount but on par with EthHub and below Week in Ethereum. However, even a quadratic matching grant of this size still raises to the next criticism: **is twittering a public good or public bad anyway**?

Traditionally, public goods of the type that Gitcoin Grants quadratic funding is trying to support were selected and funded by governments. The motivation of @antiprosynth's tweets is "aggregating Ethereum-related news, fighting information asymmetry and fine-tuning/signaling a consistent narrative for Ethereum (and ETH)": essentially, fighting the good fight against anti-Ethereum misinformation by [bitcoin maximalists](#). And, lo and behold, governments too have a rich history of [sponsoring social media participants](#) to argue on their behalf. And it seems likely that most of these governments see themselves as "fighting the good fight against anti-[X] misinformation by [Y] {extremists, imperialists, totalitarians}", just

as the Ethereum community feels a need to fight the good fight against maximalist trolls. From the inside view of each individual country (and in our case the Ethereum community) organized social media participation seems to be a clear public good (ignoring the possibility of blowback effects, which are real and important). But from the outside view of the entire world, it can be viewed as a zero-sum game.
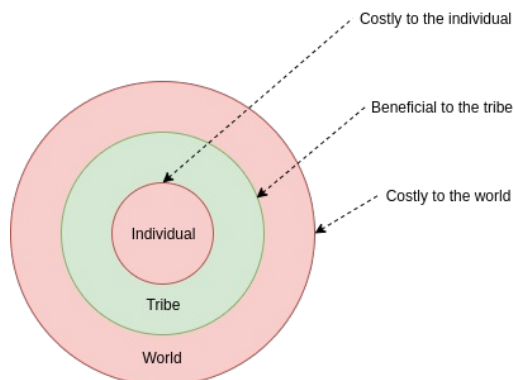
This is actually a common pattern to see in politics, and indeed there are many instances of larger-scale coordination that are precisely intended to undermine smaller-scale coordination that is seen as "good for the tribe but bad for the world": antitrust law, free trade agreements, state-level pre-emption of local zoning codes, anti-militarization agreements... the list goes on. A broad environment where public subsidies are generally viewed suspiciously also does quite a good job of limiting many kinds of malign local coordination. But as public goods become more important, and we discover better and better ways for communities to coordinate on producing them, that strategy's efficacy becomes more limited, and properly grappling with these discrepancies between what is good for the tribe and what is good for the world becomes more important.

That said, internet marketing and debate is not a zero-sum game, and there are plenty of ways to engage in internet marketing and debate that are good for the world. Internet debate in general serves to help the public learn what things are true, what things are not true, what causes to support, and what causes to oppose. Some tactics are clearly not truth-favoring, but other tactics are quite truth-favoring. Some tactics are clearly offensive, but others are defensive. And in the ethereum community, there [is](#) [widespread](#) [sentiment](#) that there is not enough resources going into marketing of *some* kind, and I personally agree with this sentiment.

What kind of marketing is positive-sum (good for tribe and good for world) and what kind of marketing is zero-sum (good for tribe but bad for world) is another question, and one that's worth the community debating. I naturally hope that the Ethereum community continues to value maintaining a moral high ground. Regarding the case of @antiprosynth himself, I cannot find any tactics that I would classify as bad-for-world, especially when compared to outright misinformation ("it's impossible to run a full node") that we often see used against Ethereum - but I am pro-ethereum and hence biased, hence the need to be careful.

Universal mechanisms, particular goals

The story has another plot twist, which reveals yet another feature (or bug?) or quadratic funding. Quadratic funding [was originally described](#) as "Formal Rules for a Society Neutral among Communities", the intention being to use it at a very large, potentially even global, scale. Anyone can participate as a project or as a participant, and projects that support public goods that are good for *any* "public" would be supported. In the case of Gitcoin Grants, however, the matching funds are coming from Ethereum organizations, and so there is an expectation that the system is there to support Ethereum projects. But there is nothing in the rules of quadratic funding that privileges Ethereum projects and prevents, say, Ethereum Classic projects from seeking funding using the same platform! And, of course, this is exactly what happened:

So now the result is, $24 of funding from Ethereum organizations will be going toward supporting an Ethereum Classic promoter's twitter activity. To give people outside of the crypto space a feeling for what this is like, imagine the USA holding a quadratic funding raise, using government funding to match donations, and the result is that some of the funding goes to someone explicitly planning to use the money to talk on Twitter about how great Russia is (or vice versa). The matching funds are coming from Ethereum sources, and there's an implied expectation that the funds should support Ethereum, but nothing actually prevents, or even discourages, non-Ethereum projects from organizing to get a share of the matched funds on the platform!

Solutions

There are two solutions to these problems. One is to modify the quadratic funding mechanism to support negative votes in addition to positive votes. The mathematical theory behind quadratic voting already implies that it is the "right thing" to do to allow such a possibility (every positive number has a negative square root as well as a positive square root). On the other hand, there are social concerns that allowing for negative voting would cause more animosity and lead to other kinds of harms. After all, mob mentality is at its worst when it is against something rather than for something. Hence, it's my view that it's not certain that allowing negative contributions will work out well, but there is enough evidence that it might that it is definitely worth trying out in a future round.

The second solution is to use two separate mechanisms for identifying relative goodness of good projects and for screening out bad projects. For example, one could use a challenge mechanism followed by a majority ETH coin vote, or even at first just a centralized appointed board, to screen out bad projects, and then use quadratic funding as before to choose between good projects. This is less mathematically elegant, but it would solve the problem, and it would at the same time provide an opportunity to mix in a separate mechanism to ensure that chosen projects benefit Ethereum specifically.

But even if we adopt the first solution, defining boundaries for the quadratic funding itself may also be a good idea. There is intellectual precedent for this. In Elinor Ostrom's eight principles for governing the commons, defining clear boundaries about who has the right to access the commons is the first one. Without clear boundaries, Ostrom writes, "local appropriators face the risk that any benefits they produce by their efforts will be reaped by others who have not contributed to those efforts." In the case of Gitcoin Grants quadratic funding, one possibility would be to set the maximum matching coefficient for any pair of users to be proportional to the geometric average of their ETH holdings, using that as a proxy for measuring membership in the Ethereum community (note that this avoids being plutocratic because 1000 users with 1 ETH each would have a maximum matching of $\approx k * 500,000$ ETH, whereas 2 users with 500 ETH each would only have a maximum matching of $k * 1,000$ ETH).

Collusion

Another issue that came to the forefront this round was the issue of collusion. The math behind quadratic funding, which compensates for tragedies of the commons by magnifying individual contributions based on the total number and size of other contributions to the same project, only works if there is an actual tragedy of the commons limiting natural donations to the project. If there is a "quid pro quo", where people get something individually in exchange for their contributions, the mechanism can easily over-compensate. The long-run solution to this is something like MACI, a cryptographic system that ensures that contributors have no way to prove their contributions to third parties, so any such collusion would have to be done by honor system. In the short run, however, the rules and enforcement has not yet been set, and this has led to vigorous debate about what kinds of quid pro quo are legitimate:

[Update 2020.01.29: the above was [ultimately a result of a miscommunication from Gitcoin](); a member of the Gitcoin team had okayed Richard Burton's proposal to give rewards to donors without realizing the implications. So Richard himself is blameless here; though the broader point that we underestimated the need for explicit guidance about what kinds of quid pro quos are acceptable is very much real.]

Currently, the position is that [quid pro quos are disallowed](), though there is a more nuanced feeling that informal social quid pro quos ("thank yous" of different forms) are okay, whereas formal and especially monetary or product rewards are a no-no. This seems like a reasonable approach, though it does put Gitcoin further into the uncomfortable position of being a central arbiter, compromising [credible neutrality]() somewhat. One positive byproduct of this whole discussion is that it has led to much more awareness in the Ethereum community of what actually is a public good (as opposed to a "private good" or a "club good"), and more generally brought public goods much further into the public discourse.

Conclusions

Whereas round 3 was the first round with enough participants to have any kind of interesting effects, round 4 felt like a true "coming-out party" for the cause of decentralized public goods funding. The round attracted a large amount of attention from the community, and even from outside actors such as the Bitcoin community. It is part of a broader trend in the last few months where public goods funding has become a [dominant part]() of the crypto community discourse. Along with this, we have also seen much more [discussion]() of [strategies]() about long-term sources of funding for quadratic matching pools of larger sizes.

Discussions about funding will be important going forward: donations from large Ethereum organizations are enough to sustain quadratic matching at its current scale, but not enough to allow it to grow much further, to the point where we can have hundreds of quadratic freelancers instead of about five. At those scales, sources of funding for Ethereum public goods must rely on network effect lockin to some extent, or else they will have little more staying power than individual donations, but there are strong reasons not to embed these funding sources too deeply into Ethereum (eg. into the protocol itself, a la [the recent BCH proposal]()), to avoid risking the protocol's neutrality.

Approaches based on capturing transaction fees at layer 2 are surprisingly viable: currently, there are about $50,000-100,000 per day (~$18-35m per year) of transaction fees happening on Ethereum, roughly equal to the entire budget of the Ethereum Foundation. And there is evidence that [miner-extractable value]() is even higher. There are all discussions that we need to have, and challenges that we need to address, if we want the Ethereum community to be a leader in implementing decentralized, credibly neutral and market-based solutions to public goods funding challenges.

Base Layers And Functionality Escape Velocity

2019 Dec 26

One common strand of thinking in blockchain land goes as follows: blockchains should be maximally simple, because they are a piece of infrastructure that is difficult to change and would lead to great harms if it breaks, and more complex functionality should be built on top, in the form of layer 2 protocols: state channels, Plasma, rollup, and so forth. Layer 2 should be the site of ongoing innovation, layer 1 should be the site of stability and maintenance, with large changes only in emergencies (eg. a one-time set of serious breaking changes to prevent the base protocol's cryptography from falling to quantum computers would be okay).

This kind of layer separation is a very nice idea, and in the long term I strongly support this idea. However, this kind of thinking misses an important point: while layer 1 cannot be *too* powerful, as greater power implies greater complexity and hence greater brittleness, layer 1 must also be *powerful enough* for the layer 2 protocols-on-top that people want to build to actually be possible in the first place. Once a layer 1 protocol has achieved a certain level of functionality, which I will term "functionality escape velocity", then yes, you can do everything else on top without further changing the base. But if layer 1 is not powerful enough, then you can talk about filling in the gap with layer 2 systems, but the reality is that there is no way to actually build those systems, without reintroducing a whole set of trust assumptions that the layer 1 was trying to get away from. This post will talk about some of what this minimal functionality that constitutes "functionality escape velocity" is.

A programming language

It must be possible to execute custom user-generated scripts on-chain. This programming language can be simple, and actually does not need to be high-performance, but it needs to at least have the level of functionality required to be able to verify arbitrary things that might need to be verified. This is important because the layer 2 protocols that are going to be built on top need to have some kind of verification logic, and this verification logic must be executed by the blockchain somehow.

You may have heard of Turing completeness; the "layman's intuition" for the term being that if a programming language is Turing complete then it can do anything that a computer theoretically could do. Any program in one Turing-complete language can be translated into an equivalent program in any other Turing-complete language. However, it turns out that we only need something slightly lighter: it's okay to restrict to programs without loops, or programs which are guaranteed to terminate in a specific number of steps.

Rich Statefulness

It doesn't just matter that a programming language *exists*, it also matters precisely how that programming language is integrated into the blockchain. Among the more constricted ways that a language could be integrated is if it is used for pure transaction verification: when you send coins to some address, that address represents a computer program P which would be used to verify a transaction that sends coins *from* that address. That is, if you send a transaction whose hash is h, then you would supply a signature S, and the blockchain would run $P(h, S)$, and if that outputs TRUE then the transaction is valid. Often, P is a verifier for a cryptographic signature scheme, but it could do more complex operations. Note particularly that in this model P *does not have access* to the destination of the transaction.

However, this "pure function" approach is not enough. This is because this pure function-based approach is not powerful enough to implement many kinds of layer 2 protocols that people actually want to implement. It can do channels (and channel-based systems like the Lightning Network), but it cannot implement other scaling techniques with stronger properties, it cannot be used to bootstrap systems that do have more complicated notions of state, and so forth.

To give a simple example of what the pure function paradigm cannot do, consider a savings account with the following feature: there is a cryptographic key k which can initiate a withdrawal, and if a withdrawal is initiated, within the next 24 hours that same key k can cancel the withdrawal. If a withdrawal remains uncancelled within 24 hours, then anyone can "poke" the account to finalize that

withdrawal. The goal is that if the key is stolen, the account holder can prevent the thief from withdrawing the funds. The thief could of course prevent the legitimate owner from getting the funds, but the attack would not be profitable for the thief and so they would probably not bother with it (see [the original paper](#) for an explanation of this technique).

Unfortunately this technique cannot be implemented with just pure functions. The problem is this: there needs to be some way to move coins from a "normal" state to an "awaiting withdrawal" state. But the program P does not have access to the destination! Hence, any transaction that could authorize moving the coins to an awaiting withdrawal state could also authorize just stealing those coins immediately; P can't tell the difference. The ability to change the state of coins, without completely setting them free, is important to many kinds of applications, including layer 2 protocols. Plasma itself fits into this "authorize, finalize, cancel" paradigm: an exit from Plasma must be approved, then there is a 7 day challenge period, and within that challenge period the exit could be cancelled if the right evidence is provided. Rollup also needs this property: coins inside a rollup must be controlled by a program that keeps track of a state root R, and changes from R to R' if some verifier P(R, R', data) returns TRUE - but it only changes the state to R' in that case, it does not set the coins free.

This ability to authorize state changes without completely setting all coins in an account free, is what I mean by "rich statefulness". It can be implemented in many ways, some UTXO-based, but without it a blockchain is not powerful enough to implement most layer 2 protocols, without including trust assumptions (eg. a set of functionaries who are collectively trusted to execute those richly-stateful programs).

Note: yes, I know that if P has access to h then you can just include the destination address as part of S and check it against h, and restrict state changes that way. But it is possible to have a programming language that is too resource-limited or otherwise restricted to actually do this; and surprisingly this often actually is the case in blockchain scripting languages.

Sufficient data scalability and low latency

It turns out that plasma and channels, and other layer 2 protocols that are fully off-chain have some fundamental weaknesses that prevent them from fully replicating the capabilities of layer 1. I go into this in detail [here](#); the summary is that these protocols need to have a way of adjudicating situations where some parties maliciously fail to provide data that they promised to provide, and because data publication is not globally verifiable (you don't know when data was published unless you already downloaded it yourself) these adjudication games are not game-theoretically stable. Channels and Plasma cleverly get around this instability by adding additional assumptions, particularly assuming that for every piece of state, there is a single actor that is interested in that state not being incorrectly modified (usually because it represents coins that they own) and so can be trusted to fight on its behalf. However, this is far from general-purpose; systems like [Uniswap](#), for example, include a large "central" contract that is not owned by anyone, and so they cannot effectively be protected by this paradigm.

There is one way to get around this, which is layer 2 protocols that publish very small amounts of data on-chain, but do computation entirely off-chain. If data is guaranteed to be available, then computation being done off-chain is okay, because games for adjudicating who did computation correctly and who did it incorrectly *are* game-theoretically stable (or could be replaced entirely by [SNARKs](#) or [STARKs](#)). This is the logic behind [ZK rollup](#) and [optimistic rollup](#). If a blockchain allows for the publication and guarantees the availability of a reasonably large amount of data, even if its capacity for *computation* remains very limited, then the blockchain can support these layer-2 protocols and achieve a high level of scalability *and* functionality.

Just how much data does the blockchain need to be able to process and guarantee? Well, it depends on what TPS you want. With a rollup, you can compress most activity to ~10-20 bytes per transaction, so 1 kB/sec gives you 50-100 TPS, 1 MB/sec gives you 50,000-100,000 TPS, and so forth. Fortunately, internet bandwidth [continues to grow quickly](#), and does not seem to be slowing down the way Moore's law for computation is, so increasing scaling for data without increasing computational load is quite a viable path for blockchains to take!

Note also that it is not just data capacity that matters, it is also data latency (ie. having low block times). Layer 2 protocols like rollup (or for that matter Plasma) only give any guarantees of security when the data actually is published to chain; hence, the time it takes for data to be reliably included (ideally "finalized") on chain is the time that it takes between when Alice sends Bob a payment and Bob can be confident that this payment will be included. The block time of the base layer sets the latency for anything whose confirmation depends things being included in the base layer. This could be worked around with on-chain security deposits, aka "bonds", at the cost of high capital

inefficiency, but such an approach is inherently imperfect because a malicious actor could trick an unlimited number of different people by sacrificing one deposit.

Conclusions

"Keep layer 1 simple, make up for it on layer 2" is NOT a universal answer to blockchain scalability and functionality problems, because it fails to take into account that layer 1 blockchains themselves must have a sufficient level of scalability and functionality for this "building on top" to actually be possible (unless your so-called "layer 2 protocols" are just trusted intermediaries). However, it is true that beyond a certain point, any layer 1 functionality *can* be replicated on layer 2, and in many cases it's a good idea to do this to improve upgradeability. Hence, we need [layer 1 development in parallel with layer 2 development in the short term, and more focus on layer 2 in the long term](#).

Christmas Special

2019 Dec 24

Since it's Christmas time now, and we're theoretically supposed to be enjoying ourselves and spending time with our families instead of waging endless holy wars on Twitter, this blog post will offer some games that you can play with your friends that will help you have fun *and* at the same time understand some spooky mathematical concepts!

1.58 dimensional chess

This is a variant of chess where the board is set up like this:
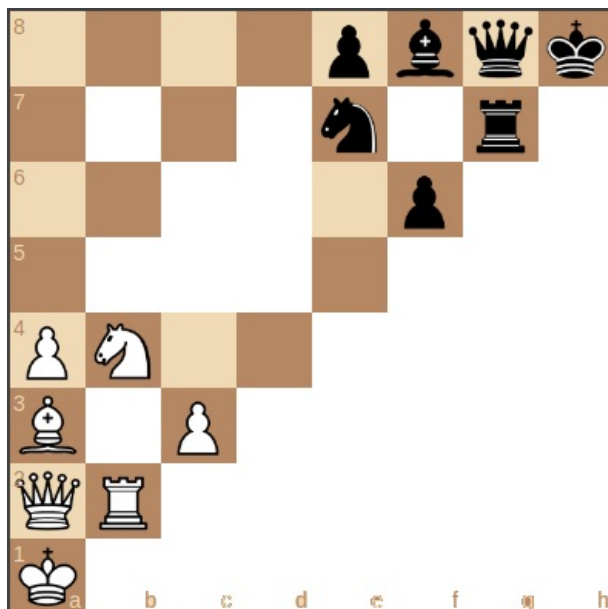
The board is still a normal 8x8 board, but there are only 27 open squares. The other 37 squares should be covered up by checkers or Go pieces or anything else to denote that they are inaccessible. The rules are the same as chess, with a few exceptions:

- White pawns move up, black pawns move left. White pawns take going left-and-up or right-and-up, black pawns take going left-and-down or left-and-up. White pawns promote upon reaching the top, black pawns promote upon reaching the left.
- No en passant, castling, or two-step-forward pawn jumps.
- Chess pieces cannot move onto *or through* the 37 covered squares. Knights cannot move onto the 37 covered squares, but don't care what they move "through".

The game is called 1.58 dimensional chess because the 27 open squares are chosen according to a pattern based on the [Sierpinski triangle](). You start off with a single open square, and then every time you double the width, you take the shape at the end of the previous step, and copy it to the top left, top right and bottom left corners, but leave the bottom right corner inaccessible. Whereas in a one-dimensional structure, doubling the width increases the space by 2x, and in a two-dimensional structure, doubling the width increases the space by 4x ($4 = 2^2$), and in a three-dimensional structure, doubling the width increases the space by 8x ($8 = 2^3$), here doubling the width increases the space by 3x ($3 = 2^{1.58496}$), hence "1.58 dimensional" (see [Hausdorff dimension]() for details).

The game is substantially simpler and more "tractable" than full-on chess, and it's an interesting exercise in showing how in [lower-dimensional spaces]() defense becomes much easier than offense. Note that the relative value of different pieces may change here, and new kinds of endings become possible (eg. you can checkmate with just a bishop).

3 dimensional tic tac toe
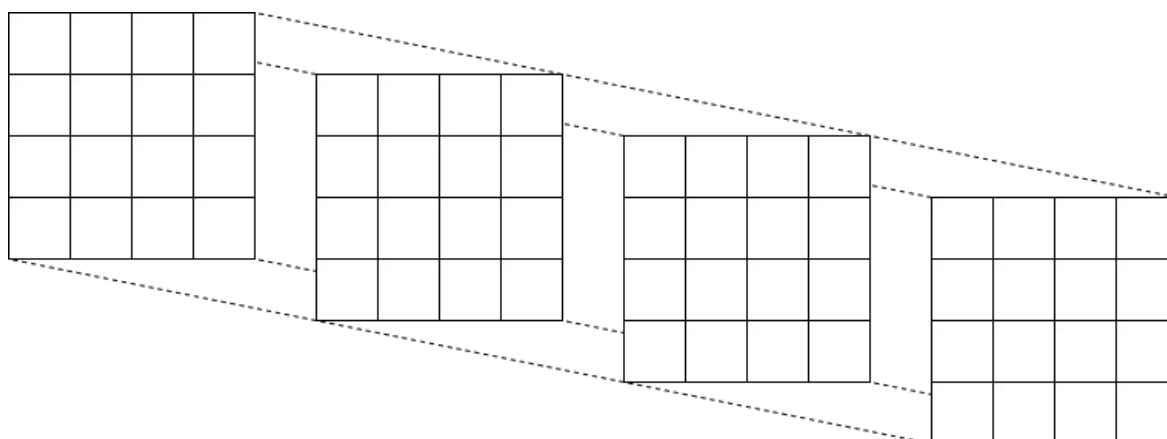
The goal here is to get 4 in a straight line, where the line can go in any direction, along an axis or diagonal, including between planes. For example in this configuration X wins:
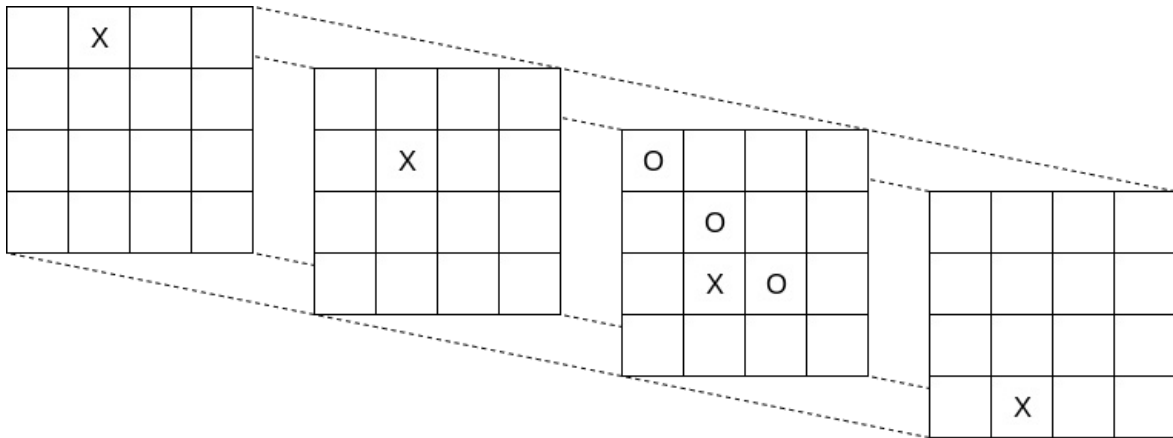
It's considerably harder than [traditional 2D tic tac toe](), and hopefully much more fun!

Modular tic-tac-toe

Here, we go back down to having two dimensions, except we allow lines to wrap around:

X wins

Note that we allow diagonal lines with any slope, as long as they pass through all four points. Particularly, this means that lines with slope +/- 2 and +/- 1/2 are admissible:

Mathematically, the board can be interpreted as a 2-dimensional vector space over [integers modulo 4](), and the goal being to fill in a line that passes through four points over this space. Note that there exists at least one line passing through any two points.

Tic tac toe over the 4-element binary field

x+1

x

1

0

　　　　0　　1　　x　　x+1

Here, we have the same concept as above, except we use an even spookier mathematical structure, the 4-element field of polynomials over Z_2 modulo $x^2 + x + 1$. This structure has pretty much no reasonable geometric interpretation, so I'll just give you the addition and multiplication tables:

Addition

	0	1	x	x+1
x+1	x+1	x	1	0
x	x	x+1	0	1
1	1	0	x+1	x
0	0	1	x	x+1

Multiplication

	0	1	x	x+1
x+1	0	x+1	1	x
x	0	x	x+1	1
1	0	1	x	x+1
0	0	0	0	0

OK fine, here are all possible lines, excluding the horizontal and the vertical lines (which are also admissible) for brevity:
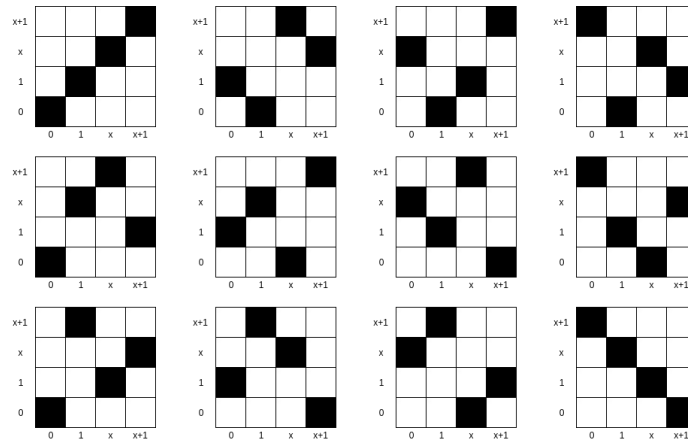
The lack of geometric interpretation does make the game harder to play; you pretty much have to memorize the twenty winning combinations, though note that they are *basically* rotations and reflections of the same four basic shapes (axial line, diagonal line, diagonal line starting in the middle, that weird thing that doesn't look like a line).

Now play 1.77 dimensional connect four. I dare you.
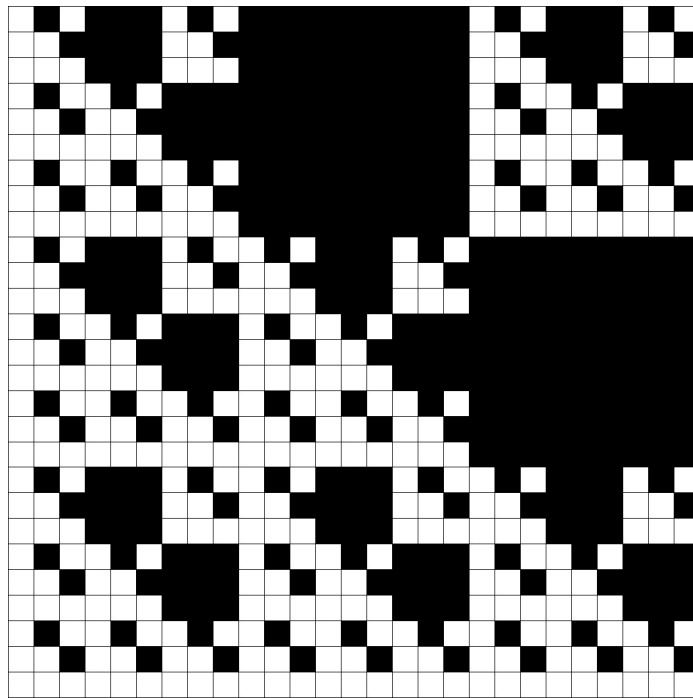
Modular poker

Everyone is dealt five (you can use whatever variant poker rules you want here in terms of how these cards are dealt and whether or not players have the right to swap cards out). The cards are interpreted as: jack = 11, queen = 12, king = 0, ace = 1. A hand is stronger than another hand, if it contains a longer sequence, with any constant difference between consecutive cards (allowing wraparound), than the other hand.

Mathametically, this can be represented as, a hand is stronger if the player can come up with a line $L(x) = mx+b$ such that they have cards for the numbers $L(0)$, $L(1)$ ... $L(k)$ for the highest k.
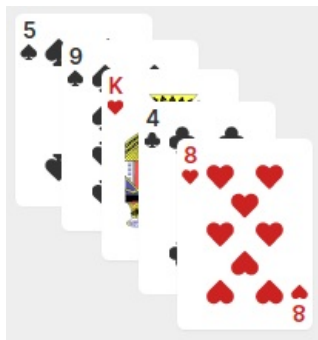
Example of a full five-card winning hand. y = 4x + 5.

To break ties between equal maximum-length sequences, count the number of distinct length-three sequences they have; the hand with more distinct length-three sequences wins.

This hand has four length-three sequences: K 2 4, K 4 8, 2 3 4, 3 8 K. This is rare.

Only consider lines of length three or higher. If a hand has three or more of the same denomination, that counts as a sequence, but if a hand has two of the same denomination, any sequences passing through that denomination only count as one sequence.
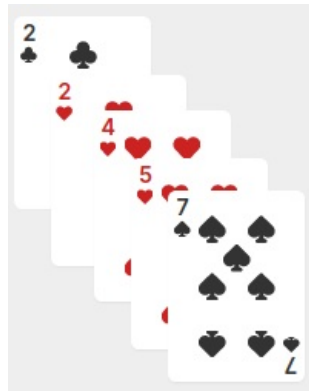
This hand has no length-three sequences.

If two hands are completely tied, the hand with the higher highest card (using J = 11, Q = 12, K = 0, A = 1 as above) wins.

Enjoy!

Quadratic Payments: A Primer

Special thanks to Karl Floersch and Jinglan Wang for feedback

If you follow applied mechanism design or decentralized governance at all, you may have recently heard one of a few buzzwords: quadratic voting, quadratic funding and quadratic attention purchase. These ideas have been gaining popularity rapidly over the last few years, and small-scale tests have already been deployed: the Taiwanese presidential hackathon used quadratic voting to vote on winning projects, Gitcoin Grants used quadratic funding to fund public goods in the Ethereum ecosystem, and the Colorado Democratic party also experimented with quadratic voting to determine their party platform.

To the proponents of these voting schemes, this is not just another slight improvement to what exists. Rather, it's an initial foray into a fundamentally new class of social technology which, has the potential to overturn how we make many public decisions, large and small. The ultimate effect of these schemes rolled out in their full form *could be as deeply transformative as the industrial-era advent of mostly-free markets and constitutional democracy*. But now, you may be thinking: "These are large promises. What do these new governance technologies have that justifies such claims?"

Private goods, private markets...

To understand what is going on, let us first consider an existing social technology: money, and property rights - the invisible social technology that generally hides behind money. Money and private property are extremely powerful social technologies, for all the reasons classical economists have been stating for over a hundred years. If Bob is producing apples, and Alice wants to buy apples, we can economically model the interaction between the two, and the results *seem to make sense*:
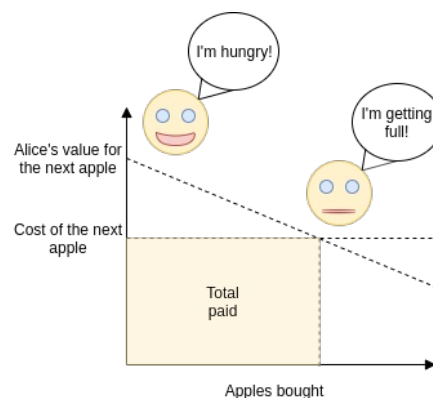
Alice keeps buying apples until the marginal value of the next apple to her is less than the cost of producing it, which is pretty much exactly the optimal thing that could happen. This is all formalized in results such as the "fundamental theorems of welfare economics". Now, those of you who have learned some economics may be screaming, but what about imperfect competition? Asymmetric information? Economic inequality? Public goods? Externalities? Many activities in the real world, including those that are key to the progress of human civilization, benefit (or harm) many people in complicated ways. These activities and the consequences that arise from them often cannot be neatly decomposed into sequences of distinct trades between two parties.

But since when do we expect a single package of technologies to solve every problem anyway? "What about oceans?" isn't an argument against *cars*, it's an argument against *car maximalism*, the position that we need cars and nothing else. Much like how private property and markets deal with private goods, can we try to use economic means to deduce what kind of social technologies would work well for encouraging production of the public goods that we need?

... Public goods, public markets

Private goods (eg. apples) and public goods (eg. public parks, air quality, scientific research, this article...) are different in some key ways. When we are talking about private goods, production for multiple people (eg. the same farmer makes apples for both Alice and Bob) can be decomposed into (i) the farmer making some apples for Alice, and (ii) the farmer making some other apples for Bob. If Alice wants apples but Bob does not, then the farmer makes Alice's apples, collects payment from Alice, and leaves Bob alone. Even complex collaborations (the "I, Pencil" essay popular in libertarian circles comes to mind) can be decomposed into a series of such interactions. When we are talking about public goods, however, *this kind of decomposition is not possible*. When I write this blog article, it can be read by both Alice and Bob (and everyone else). I *could* put it behind a paywall, but if it's popular enough it will inevitably get mirrored on third-party sites, and paywalls are in any case annoying and not very effective. Furthermore, making an article available to ten people is not ten times cheaper than making the article available to a hundred people; rather, *the cost is exactly the same*. So I either produce the article for everyone, or I do not produce it for anyone at all.

So here comes the challenge: how do we aggregate together people's preferences? Some private and public goods are

worth producing, others are not. In the case of private goods, the question is easy, because we can just decompose it into a series of decisions for each individual. Whatever amount each person is willing to pay for, that much gets produced for them; the economics is not especially complex. In the case of public goods, however, you cannot "decompose", and so we need to add up people's preferences in a different way.

First of all, let's see what happens if we just put up a plain old regular market: I offer to write an article as long as at least $1000 of money gets donated to me (fun fact: I literally did this back in 2011). Every dollar donated increases the probability that the goal will be reached and the article will be published; let us call this "marginal probability" p. At a cost of $k, you can increase the probability that the article will be published by k * p (though eventually the gains will decrease as the probability approaches 100%). Let's say to you personally, the article being published is worth $V. Would you donate? Well, donating a dollar increases the probability it will be published by p, and so gives you an expected $p * V of value. If p * V > 1, you donate, and quite a lot, and if p * V < 1 you don't donate at all.

Phrased less mathematically, either you value the article enough (and/or are rich enough) to pay, and if that's the case it's in your interest to keep paying (and influencing) quite a lot, or you don't value the article enough and you contribute nothing. Hence, the only blog articles that get published would be articles where some single person is willing to basically pay for it themselves (in my experiment in 2011, this prediction was experimentally verified: in most rounds, over half of the total contribution came from a single donor).
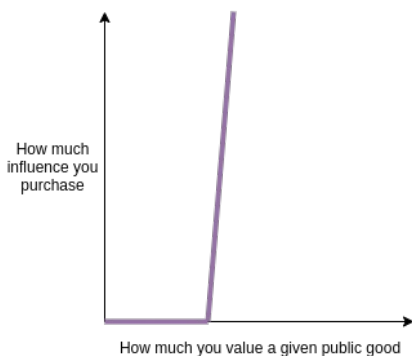
Note that *this reasoning applies for any kind of mechanism that involves "buying influence" over matters of public concern*. This includes paying for public goods, shareholder voting in corporations, public advertising, bribing politicians, and much more. The little guy has too little influence (not quite zero, because in the real world things like altruism exist) and the big guy has too much. If you had an intuition that markets work great for buying apples, but money is corrupting in "the public sphere", this is basically a simplified mathematical model that shows why.

We can also consider a different mechanism: one-person-one-vote. Let's say you can either vote that I deserve a reward for writing this article, or you can vote that I don't, and my reward is proportional to the number of votes in my favor. We can interpret this as follows: your first "contribution" costs only a small amount of effort, so you'll support an article if you care about it enough, but after that point there is no more room to contribute further; your second contribution "costs" infinity.
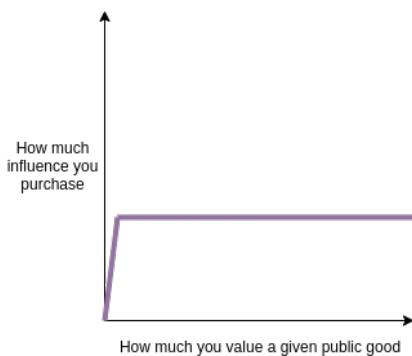
Now, you might notice that neither of the graphs above look quite right. The first graph over-privileges people who *care a lot* (or are wealthy), the second graph over-privileges people who *care only a little*, which is also a problem. The single sheep's desire to live is more important than the two wolves' desire to have a tasty dinner.

But what do we actually want? Ultimately, we want a scheme where *how much influence you "buy" is proportional to how much you care*. In the mathematical lingo above, we want your k to be proportional to your V. But here's the problem: your V determines how much you're willing to pay for *one* unit of influence. If Alice were willing to pay $100 for the article if she had to fund it herself, then she would be willing to pay $1 for an increased 1% chance it will get written, and if Bob were only willing to pay $50 for the article then he would only be willing to pay $0.5 for the same "unit of influence".

So how do we match these two up? The answer is clever: *your n'th unit of influence costs you $n* . That is, for example, you could buy your first vote for $0.01, but then your second would cost $0.02, your third $0.03, and so forth. Suppose

you were Alice in the example above; in such a system she would keep buying units of influence until the cost of the next one got to $1, so she would buy 100 units. Bob would similarly buy until the cost got to $0.5, so he would buy 50 units. Alice's 2x higher valuation turned into 2x more units of influence purchased.

Let's draw this as a graph:
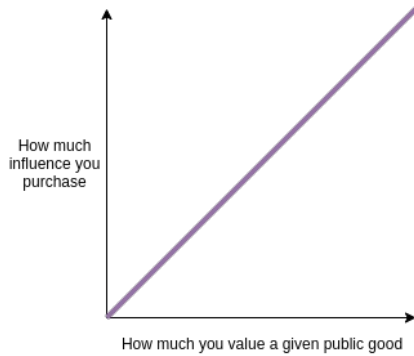
Now let's look at all three beside each other:
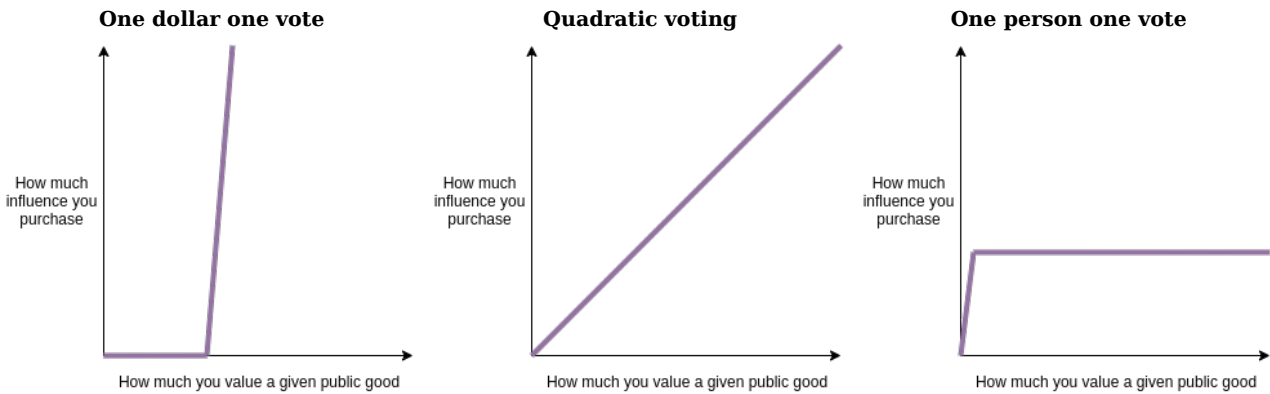
Notice that only quadratic voting has this nice property that the amount of influence you purchase is proportional to how much you care; the other two mechanisms either over-privilege concentrated interests or over-privilege diffuse interests.

Now, you might ask, where does the *quadratic* come from? Well, the *marginal* cost of the n'th vote is $n (or $0.01 * n), but the *total* cost of n votes is $\approx \frac{n^2}{2}$. You can view this geometrically as follows:
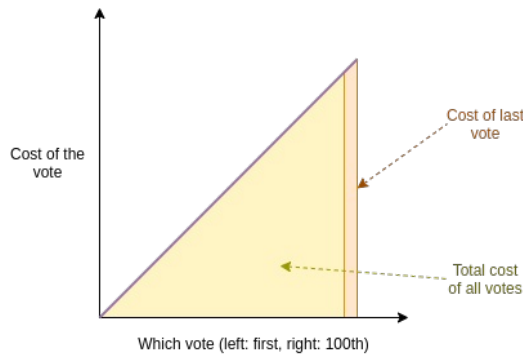
The total cost is the area of a triangle, and you probably learned in math class that area is base * height / 2. And since here base and height are proportionate, that basically means that total cost is proportional to number of votes squared - hence, "quadratic". But honestly it's easier to think "your n'th unit of influence costs $n".

Finally, you might notice that above I've been vague about what "one unit of influence" actually means. This is deliberate; it can mean different things in different contexts, and the different "flavors" of quadratic payments reflect these different perspectives.

Quadratic Voting

See also the original paper: https://papers.ssrn.com/sol3/papers.cfm?abstract%5fid=2003531

Let us begin by exploring the first "flavor" of quadratic payments: quadratic voting. Imagine that some organization is

trying to choose between two choices for some decision that affects all of its members. For example, this could be a company or a nonprofit deciding which part of town to make a new office in, or a government deciding whether or not to implement some policy, or an internet forum deciding whether or not its rules should allow discussion of cryptocurrency prices. Within the context of the organization, the choice made is a public good (or public bad, depending on whom you talk to): everyone "consumes" the results of the same decision, they just have different opinions about how much they like the result.

This seems like a perfect target for quadratic voting. The goal is that option A gets chosen if in total people like A more, and option B gets chosen if in total people like B more. With simple voting ("one person one vote"), the distinction between stronger vs weaker preferences gets ignored, so on issues where one side is of very high value to a few people and the other side is of low value to more people, simple voting is likely to give wrong answers. With a private-goods market mechanism where people can buy as many votes as they want at the same price per vote, the individual with the strongest preference (or the wealthiest) carries everything. Quadratic voting, where you can make n votes in either direction at a cost of n^2, is right in the middle between these two extremes, and creates the perfect balance.
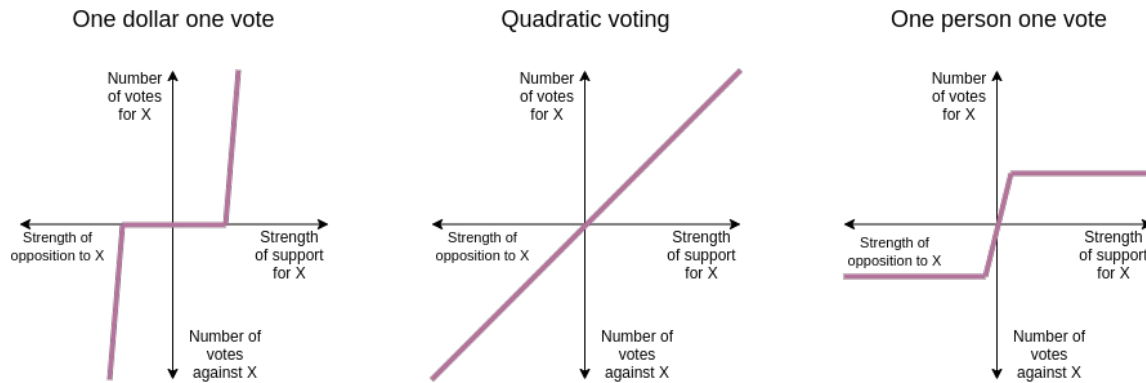
Note that in the voting case, we're deciding two options, so different people will favor A over B or B over A; hence, unlike the graphs we saw earlier that start from zero, here voting and preference can both be positive or negative (which option is considered positive and which is negative doesn't matter; the math works out the same way)

As shown above, because the n'th vote has a cost of n, the number of votes you make is proportional to how much you value one unit of influence over the decision (the value of the decision multiplied by the probability that one vote will tip the result), and hence proportional to how much you care about A being chosen over B or vice versa. Hence, we once again have this nice clean "preference adding" effect.

We can extend quadratic voting in multiple ways. First, we can allow voting between more than two options. While traditional voting schemes inevitably fall prey to various kinds of "strategic voting" issues because of Arrow's theorem and Duverger's law, quadratic voting continues to be optimal in contexts with more than two choices.

> **The intuitive argument for those interested**: suppose there are established candidates A and B and new candidate C. Some people favor C > A > B but others C > B > A. in a regular vote, if both sides think C stands no chance, they decide may as well vote their preference between A and B, so C gets no votes, and C's failure becomes a self-fulfilling prophecy. In quadratic voting the former group would vote [A +10, B -10, C +1] and the latter [A -10, B +10, C +1], so the A and B votes cancel out and C's popularity shines through.

Second, we can look not just at voting between discrete options, but also at voting on the setting of a thermostat: anyone can push the thermostat up or down by 0.01 degrees n times by paying a cost of n^2.

Plot twist: the side wanting it colder only wins when they convince the other side that "C" stands for "caliente".

Quadratic funding

*See also the original paper: https://papers.ssrn.com/sol3/papers.cfm?abstract%5fid=3243656*

Quadratic voting is optimal when you need to make some fixed number of collective decisions. But one weakness of quadratic voting is that it doesn't come with a built-in mechanism for deciding what goes on the ballot in the first place.

Proposing votes is potentially a source of considerable power if not handled with care: a malicious actor in control of it can repeatedly propose some decision that a majority weakly approves of and a minority strongly disapproves of, and keep proposing it until the minority runs out of voting tokens (if you do the math you'll see that the minority would burn through tokens much faster than the majority). Let's consider a flavor of quadratic payments that does not run into this issue, and makes the choice of decisions itself endogenous (ie. part of the mechanism itself). In this case, the mechanism is specialized for one particular use case: individual provision of public goods.

Let us consider an example where someone is looking to produce a public good (eg. a developer writing an open source software program), and we want to figure out whether or not this program is worth funding. But instead of just thinking about one single public good, let's create a mechanism where *anyone* can raise funds for what they claim to be a public good project. Anyone can make a contribution to any project; a mechanism keeps track of these contributions and then at the end of some period of time the mechanism calculates a payment to each project. The way that this payment is calculated is as follows: for any given project, take the square root of each contributor's contribution, add these values together, and take the square of the result. Or in math speak:

$$(\sum_{i=1}^n \sqrt{c_i})^2$$

If that sounds complicated, here it is graphically:

Green area: sum of contributions

Yellow area: subsidy if k=1
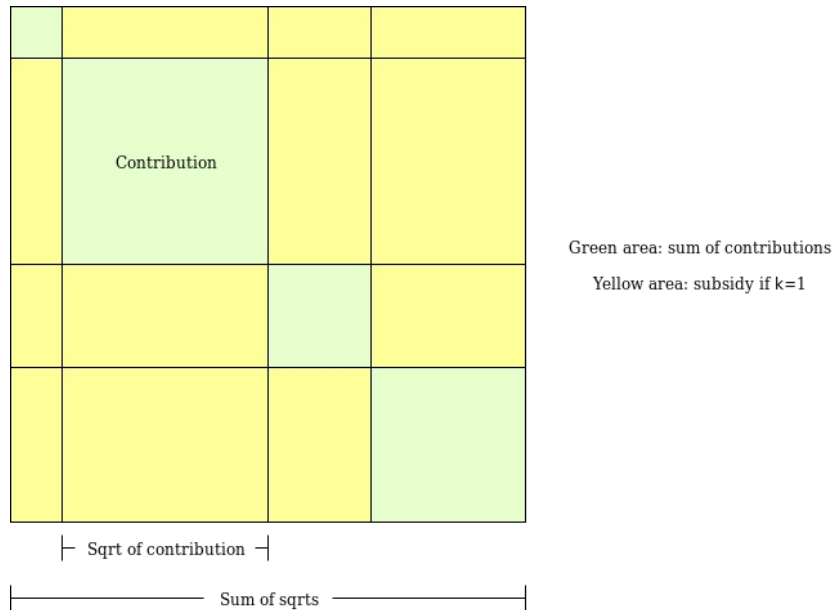
In any case where there is more than one contributor, the computed payment is greater than the raw sum of contributions; the difference comes out of a central subsidy pool (eg. if ten people each donate $1, then the sum-of-square-roots is $10, and the square of that is $100, so the subsidy is $90). Note that if the subsidy pool is not big enough to make the full required payment to every project, we can just divide the subsidies proportionately by whatever constant makes the totals add up to the subsidy pool's budget; **you can prove that this solves the tragedy-of-the-commons problem as well as you can with that subsidy budget**.

There are two ways to intuitively interpret this formula. First, one can look at it through the "fixing market failure" lens, a surgical fix to the [tragedy of the commons](#) problem. In any situation where Alice contributes to a project and Bob also contributes to that same project, Alice is making a contribution to something that is valuable not only to herself, but also to Bob. When deciding *how much to contribute*, Alice was only taking into account the benefit to herself, not Bob, whom she most likely does not even know. The quadratic funding mechanism adds a subsidy to compensate for this effect, determining how much Alice "would have" contributed if she also took into account the benefit her contribution brings to Bob. Furthermore, we can separately calculate the subsidy for each pair of people (nb. if there are N people there are N * (N-1) / 2 pairs), and add up all of these subsidies together, and give Bob the combined subsidy from all pairs. And it turns out that this gives exactly the quadratic funding formula.

Second, one can look at the formula through a quadratic voting lens. We interpret the quadratic funding as being *a special case* of quadratic voting, where the contributors to a project are voting for that project and there is one imaginary participant voting against it: the subsidy pool. Every "project" is a motion to take money from the subsidy pool and give it to that project's creator. Everyone sending c_i of funds is making $\sqrt{c_i}$ votes, so there's a total of $\sum_{i=1}^n \sqrt{c_i}$ votes in favor of the motion. To kill the motion, the subsidy pool would need to make more than $\sum_{i=1}^n \sqrt{c_i}$ votes against it, which would cost it more than $(\sum_{i=1}^n \sqrt{c_i})^2$. Hence, $(\sum_{i=1}^n \sqrt{c_i})^2$ is the maximum transfer from the subsidy pool to the project that the subsidy pool would not vote to stop.

Quadratic funding is starting to be explored as a mechanism for funding public goods already; [Gitcoin grants](#) for funding public goods in the Ethereum ecosystem is currently the biggest example, and the most recent round led to results that, in my own view, did a quite good job of making a fair allocation to support projects that the community deems valuable.
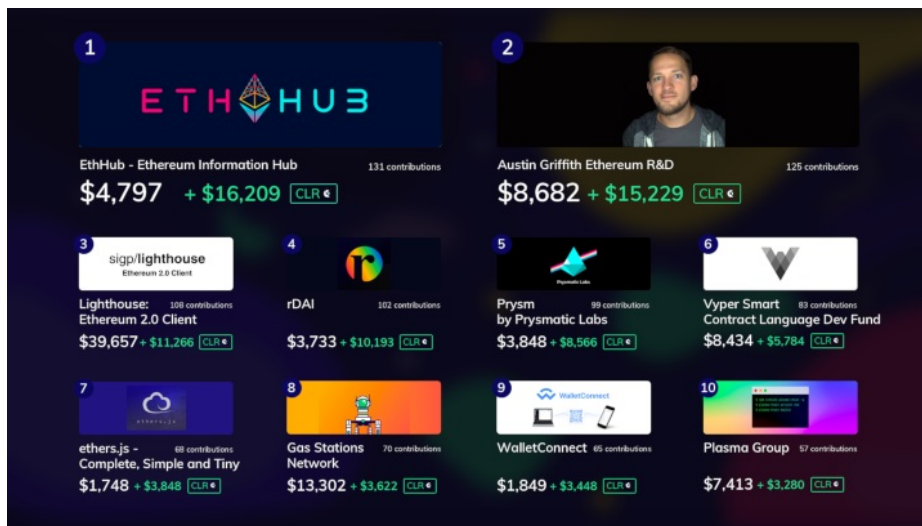
Numbers in white are raw contribution totals; numbers in green are the extra subsidies.

Quadratic attention payments

See also the original post: https://kortina.nyc/essays/speech-is-free-distribution-is-not-a-tax-on-the-purchase-of-human-attention-and-political-power/

One of the defining features of modern capitalism that people love to hate is ads. Our cities have ads:

Source: https://www.flickr.com/photos/argonavigo/36657795264
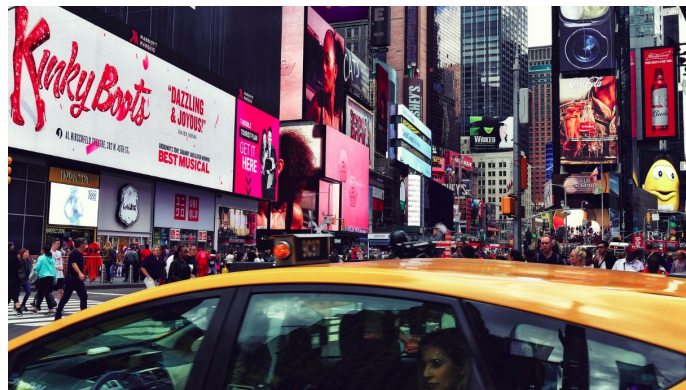
Our subway turnstiles have ads:

Source: https://commons.wikimedia.org/wiki/File:NYC,_subway_ad_on_Prince_St.jpg

Our politics are dominated by ads:

And even the rivers and the skies have ads. Now, there are some places that seem to not have this problem:

But really they just have a different kind of ads:

Now, recently there are attempts to move beyond this in some cities. And on Twitter. But let's look at the problem systematically and try to see what's going wrong. The answer is actually surprisingly simple: public advertising is the evil twin of public goods production. In the case of public goods production, there is one actor that is taking on an expenditure to produce some product, and this product benefits a large number of people. Because these people cannot effectively coordinate to pay for the public goods by themselves, we get much less public goods than we need, and the ones we do get are those favored by wealthy actors or centralized authorities. Here, there is one actor that reaps a large *benefit* from forcing other people to look at some image, and this action *harms* a large number of people. Because these people cannot effectively coordinate to buy out the slots for the ads, we get ads we don't want to see, that are favored

by... wealthy actors or centralized authorities.

So how do we solve this dark mirror image of public goods production? With a bright mirror image of quadratic funding: quadratic fees! Imagine a billboard where anyone can pay $1 to put up an ad for one minute, but if they want to do this multiple times the prices go up: $2 for the second minute, $3 for the third minute, etc. Note that you can pay to extend the lifetime of *someone else's* ad on the billboard, and this also costs you only $1 for the first minute, *even if other people already paid to extend the ad's lifetime many times*. We can once again interpret this as being a special case of quadratic voting: it's basically the same as the "voting on a thermostat" example above, but where the thermostat in question is the number of seconds an ad stays up.

This kind of payment model could be applied in cities, on websites, at conferences, or in many other contexts, if the goal is to optimize for putting up things that people want to see (or things that people want other people to see, but even here it's much more democratic than simply buying space) rather than things that wealthy people and centralized institutions want people to see.

Complexities and caveats

Perhaps the biggest challenge to consider with this concept of quadratic payments is the practical implementation issue of identity and bribery/collusion. Quadratic payments in any form require a model of identity where individuals cannot easily get as many identities as they want: if they could, then they could just keep getting new identities and keep paying $1 to influence some decision as many times as they want, and the mechanism collapses into linear vote-buying. Note that the identity system does *not* need to be airtight (in the sense of preventing multiple-identity acquisition), and indeed there are good civil-liberties reasons why identity systems probably should *not* try to be airtight. Rather, it just needs to be robust enough that manipulation is not worth the cost.

Collusion is also tricky. If we can't prevent people from selling their votes, the mechanisms once again collapse into one-dollar-one-vote. We don't just need votes to be anonymous and private (while still making the final result provable and public); **we need votes to be so private that even the person who made the vote can't prove to anyone else what they voted for**. This is difficult. Secret ballots do this well in the offline world, but secret ballots are a nineteenth century technology, far too inefficient for the sheer amount of quadratic voting and funding that we want to see in the twenty first century.

Fortunately, there are technological means that can help, combining together zero-knowledge proofs, encryption and other cryptographic technologies to achieve the precise desired set of privacy and verifiability properties. There's also proposed techniques to verify that private keys actually are in an individual's possession and not in some hardware or cryptographic system that can restrict how they use those keys. However, these techniques are all untested and require quite a bit of further work.

Another challenge is that quadratic payments, being a payment-based mechanism, continues to favor people with more money. Note that because the cost of votes is quadratic, this effect is dampened: someone with 100 times more money only has 10 times more influence, not 100 times, so the extent of the problem goes down by 90% (and even more for ultra-wealthy actors). That said, it may be desirable to mitigate this inequality of power further. This could be done either by denominating quadratic payments in a separate token of which everyone gets a fixed number of units, or giving each person an allocation of funds that can only be used for quadratic-payments use cases: this is basically Andrew Yang's "democracy dollars" proposal.

A third challenge is the "rational ignorance" and "rational irrationality" problems, which is that decentralized public decisions have the weakness that any single individual has very little effect on the outcome, and so little motivation to make sure they are supporting the decision that is best for the long term; instead, pressures such as tribal affiliation may dominate. There are many strands of philosophy that emphasize the ability of large crowds to be very wrong despite (or because of!) their size, and quadratic payments in any form do little to address this.

Quadratic payments do better at mitigating this problem than one-person-one-vote systems, and these problems can be expected to be less severe for medium-scale public goods than for large decisions that affect many millions of people, so it may not be a large challenge at first, but it's certainly an issue worth confronting. One approach is combining quadratic voting with elements of sortition. Another, potentially more long-term durable, approach is to combine quadratic voting with another economic technology that is much more specifically targeted toward rewarding the "correct contrarianism" that can dispel mass delusions: prediction markets. A simple example would be a system where quadratic funding is done *retrospectively*, so people vote on which public goods were valuable some time ago (eg. even 2 years), and projects are funded up-front by selling shares of the results of these deferred votes; by buying shares people would be both funding the projects and betting on which project would be viewed as successful in 2 years' time. There is a large design space to experiment with here.

Conclusion

As I mentioned at the beginning, quadratic payments do not solve every problem. They solve the problem of governing resources that affect large numbers of people, but they do not solve many other kinds of problems. A particularly important one is information asymmetry and low quality of information in general. For this reason, I am a fan of techniques such as prediction markets (see electionbettingodds.com for one example) to solve information-gathering problems, and many applications can be made most effective by combining different mechanisms together.

One particular cause dear to me personally is what I call "entrepreneurial public goods": public goods that in the present only a few people believe are important but in the future many more people will value. In the 19th century, contributing to abolition of slavery may have been one example; in the 21st century I can't give examples that will satisfy every reader because it's the nature of these goods that their importance will only become common knowledge later down the road, but I would point to life extension and AI risk research as two possible examples.

That said, we don't need to solve every problem today. Quadratic payments are an idea that has only become popular in the last few years; we still have not seen more than small-scale trials of quadratic voting and funding, and quadratic attention payments have not been tried at all! There is still a long way to go. But if we can get these mechanisms off the ground, there is a lot that these mechanisms have to offer!

Hard Problems in Cryptocurrency: Five Years Later

2019 Nov 22

Special thanks to Justin Drake and Jinglan Wang for feedback

In 2014, I made a post and a presentation with a list of hard problems in math, computer science and economics that I thought were important for the cryptocurrency space (as I then called it) to be able to reach maturity. In the last five years, much has changed. But exactly how much progress on what we thought then was important has been achieved? Where have we succeeded, where have we failed, and where have we changed our minds about what is important? In this post, I'll go through the 16 problems from 2014 one by one, and see just where we are today on each one. At the end, I'll include my new picks for hard problems of 2019.

The problems are broken down into three categories: (i) cryptographic, and hence expected to be solvable with purely mathematical techniques if they are to be solvable at all, (ii) consensus theory, largely improvements to proof of work and proof of stake, and (iii) economic, and hence having to do with creating structures involving incentives given to different participants, and often involving the application layer more than the protocol layer. We see significant progress in all categories, though some more than others.

Cryptographic problems

1. Blockchain Scalability

One of the largest problems facing the cryptocurrency space today is the issue of scalability ... The main concern with [oversized blockchains] is trust: if there are only a few entities capable of running full nodes, then those entities can conspire and agree to give themselves a large number of additional bitcoins, and there would be no way for other users to see for themselves that a block is invalid without processing an entire block themselves. **Problem:** create a blockchain design that maintains Bitcoin-like security guarantees, but where the maximum size of the most powerful node that needs to exist for the network to keep functioning is substantially sublinear in the number of transactions.

Status: **Great theoretical progress, pending more real-world evaluation**.

Scalability is one technical problem that we have had a huge amount of progress on theoretically. Five years ago, almost no one was thinking about sharding; now, sharding designs are commonplace. Aside from ethereum 2.0, we have OmniLedger, LazyLedger, Zilliqa and research papers seemingly coming out every month. In my own view, further progress at this point is incremental. Fundamentally, we already have a number of techniques that allow groups of validators to securely come to consensus on much more data than an individual validator can process, as well as techniques allow clients to indirectly verify the full validity and availability of blocks even under 51% attack conditions.

These are probably the most important technologies:

- **Random sampling**, allowing a small randomly selected committee to statistically stand in for the full validator set: https://github.com/ethereum/wiki/wiki/Sharding-FAQ#how-can-we-solve-the-single-shard-takeover-attack-in-an-uncoordinated-majority-model
- **Fraud proofs**, allowing individual nodes that learn of an error to broadcast its presence to everyone else: https://bitcoin.stackexchange.com/questions/49647/what-is-a-fraud-proof
- **Proofs of custody**, allowing validators to probabilistically prove that they individually downloaded and verified some piece of data: https://ethresear.ch/t/1-bit-aggregation-friendly-

custody-bonds/2236
- **Data availability proofs**, allowing clients to detect when the bodies of blocks that they have headers for [are unavailable](#): https://arxiv.org/abs/1809.09044. See also the newer [coded Merkle trees](#) proposal.

There are also other smaller developments like [Cross-shard communication via receipts](#) as well as "constant-factor" enhancements such as BLS signature aggregation.

That said, fully sharded blockchains have still not been seen in live operation (the partially sharded Zilliqa has recently started running). On the theoretical side, there are mainly disputes about details remaining, along with challenges having to do with stability of sharded networking, developer experience and mitigating risks of centralization; fundamental technical possibility no longer seems in doubt. But the challenges that *do* remain are challenges that cannot be solved by just thinking about them; only developing the system and seeing ethereum 2.0 or some similar chain running live will suffice.

2. Timestamping

Problem: create a distributed incentive-compatible system, whether it is an overlay on top of a blockchain or its own blockchain, which maintains the current time to high accuracy. All legitimate users have clocks in a normal distribution around some "real" time with standard deviation 20 seconds ... no two nodes are more than 20 seconds apart The solution is allowed to rely on an existing concept of "N nodes"; this would in practice be enforced with proof-of-stake or non-sybil tokens (see #9). The system should continuously provide a time which is within 120s (or less if possible) of the internal clock of >99% of honestly participating nodes. External systems may end up relying on this system; hence, it should remain secure against attackers controlling < 25% of nodes regardless of incentives.

Status: **Some progress**.

Ethereum has actually survived just fine with a 13-second block time and no particularly advanced timestamping technology; it uses a simple technique where a client does not accept a block whose stated timestamp is earlier than the client's local time. That said, this has not been tested under serious attacks. The recent [network-adjusted timestamps](#) proposal tries to improve on the status quo by allowing the client to determine the consensus on the time in the case where the client does not locally know the current time to high accuracy; this has not yet been tested. But in general, timestamping is not currently at the foreground of perceived research challenges; perhaps this will change once more proof of stake chains (including Ethereum 2.0 but also others) come online as real live systems and we see what the issues are.

3. Arbitrary Proof of Computation

Problem: create programs `POC_PROVE(P,I) -> (O,Q)` and `POC_VERIFY(P,O,Q) -> { 0, 1 }` such that `POC_PROVE` runs program `P` on input `I` and returns the program output `O` and a proof-of-computation `Q` and POC_VERIFY takes `P`, `O` and `Q` and outputs whether or not `Q` and `O` were legitimately produced by the `POC_PROVE` algorithm using `P`.

Status: **Great theoretical and practical progress**.

This is basically saying, build a SNARK (or STARK, or SHARK, or...). And [we've](#) [done](#) [it](#)! SNARKs are now increasingly well understood, and are even already being used in multiple blockchains today (including [tornado.cash](#) on Ethereum). And SNARKs are extremely useful, both as a privacy technology (see Zcash and tornado.cash) and as a scalability technology (see [ZK Rollup](#), [STARKDEX](#) and [STARKing erasure coded data roots](#)).

There are still challenges with efficiency; making arithmetization-friendly hash functions (see here and here for bounties for breaking proposed candidates) is a big one, and efficiently proving random memory accesses is another. Furthermore, there's the unsolved question of whether the O(n * log(n)) blowup in prover time is a fundamental limitation or if there is some way to make a succinct proof with only linear overhead as in bulletproofs (which unfortunately take linear time to verify). There are also ever-present risks that the existing schemes have bugs. In general, the problems are in the details rather than the fundamentals.

4. Code Obfuscation

The holy grail is to create an obfuscator O, such that given any program P the obfuscator can produce a second program O(P) = Q such that P and Q return the same output if given the same input and, importantly, Q reveals no information whatsoever about the internals of P. One can hide inside of Q a password, a secret encryption key, or one can simply use Q to hide the proprietary workings of the algorithm itself.

Status: **Slow progress**.

In plain English, the problem is saying that we want to come up with a way to "encrypt" a program so that the encrypted program would still give the same outputs for the same inputs, but the "internals" of the program would be hidden. An example use case for obfuscation is a program containing a private key where the program only allows the private key to sign certain messages.

A solution to code obfuscation would be very useful to blockchain protocols. The use cases are subtle, because one must deal with the possibility that an on-chain obfuscated program will be copied and run in an environment different from the chain itself, but there are many possibilities. One that personally interests me is the ability to remove the centralized operator from collusion-resistance gadgets by replacing the operator with an obfuscated program that contains some proof of work, making it very expensive to run more than once with different inputs as part of an attempt to determine individual participants' actions.

Unfortunately this continues to be a hard problem. There is continuing ongoing work in attacking the problem, one side making constructions (eg. this) that try to reduce the number of assumptions on mathematical objects that we do not know practically exist (eg. general cryptographic multilinear maps) and another side trying to make practical implementations of the desired mathematical objects. However, all of these paths are still quite far from creating something viable and known to be secure. See https://eprint.iacr.org/2019/463.pdf for a more general overview to the problem.

5. Hash-Based Cryptography

Problem: create a signature algorithm relying on no security assumption but the random oracle property of hashes that maintains 160 bits of security against classical computers (ie. 80 vs. quantum due to Grover's algorithm) with optimal size and other properties.

Status: **Some progress**.

There have been two strands of progress on this since 2014. SPHINCS, a "stateless" (meaning, using it multiple times does not require remembering information like a nonce) signature scheme, was released soon after this "hard problems" list was published, and provides a purely hash-based signature scheme of size around 41 kB. Additionally, STARKs have been developed, and one can create signatures of similar size based on them. The fact that not just signatures, but also general-purpose zero knowledge proofs, are possible with just hashes was definitely something I did not expect five years ago; I am very happy that this is the case. That said, size continues to be an issue, and ongoing progress (eg. see the very recent DEEP FRI) is continuing to reduce the size of proofs, though it looks like further progress will be incremental.

The main not-yet-solved problem with hash-based cryptography is aggregate signatures, similar to what BLS aggregation makes possible. It's known that we can just make a STARK over many Lamport signatures, but this is inefficient; a more efficient scheme would be welcome. (In case you're wondering if hash-based *public key encryption* is possible, the answer is, no, you can't do anything with more than a quadratic attack cost)

Consensus theory problems

6. ASIC-Resistant Proof of Work

One approach at solving the problem is creating a proof-of-work algorithm based on a type of computation that is very difficult to specialize ... For a more in-depth discussion on ASIC-resistant hardware, see https://blog.ethereum.org/2014/06/19/mining/.

Status: **Solved as far as we can**.

About six months after the "hard problems" list was posted, Ethereum settled on its ASIC-resistant proof of work algorithm: Ethash. Ethash is known as a memory-hard algorithm. The theory is that random-access memory in regular computers is well-optimized already and hence difficult to improve on for specialized applications. Ethash aims to achieve ASIC resistance by making memory access the dominant part of running the PoW computation. Ethash was not the first memory-hard algorithm, but it did add one innovation: it uses pseudorandom lookups over a two-level DAG, allowing for two ways of evaluating the function. First, one could compute it quickly if one has the entire (~2 GB) DAG; this is the memory-hard "fast path". Second, one can compute it much more slowly (still fast enough to check a single provided solution quickly) if one only has the top level of the DAG; this is used for block verification.

Ethash has proven remarkably successful at ASIC resistance; after three years and billions of dollars of block rewards, ASICs do exist but are at best 2-5 times more power and cost-efficient than GPUs. ProgPoW has been proposed as an alternative, but there is a growing consensus that ASIC-resistant algorithms will inevitably have a limited lifespan, and that ASIC resistance has downsides because it makes 51% attacks cheaper (eg. see the 51% attack on Ethereum Classic).

I believe that PoW algorithms that provide a medium level of ASIC resistance can be created, but such resistance is limited-term and both ASIC and non-ASIC PoW have disadvantages; in the long term the better choice for blockchain consensus is proof of stake.

7. Useful Proof of Work

making the proof of work function something which is simultaneously useful; a common candidate is something like Folding@home, an existing program where users can download software onto their computers to simulate protein folding and provide researchers with a large supply of data to help them cure diseases.

Status: **Probably not feasible, with one exception**.

The challenge with useful proof of work is that a proof of work algorithm requires many properties:

- Hard to compute
- Easy to verify
- Does not depend on large amounts of external data
- Can be efficiently computed in small "bite-sized" chunks

Unfortunately, there are not many computations that are useful that preserve all of these properties, and most computations that *do* have all of those properties and are "useful" are only "useful" for far too short a time to build a cryptocurrency around them.

However, there is one possible exception: zero-knowledge-proof generation. Zero knowledge proofs of aspects of blockchain validity (eg. data availability roots for a simple example) are difficult to

compute, and easy to verify. Furthermore, they are durably difficult to compute; if proofs of "highly structured" computation become too easy, one can simply switch to verifying a blockchain's entire state transition, which becomes extremely expensive due to the need to model the virtual machine and random memory accesses.

Zero-knowledge proofs of blockchain validity provide great value to users of the blockchain, as they can substitute the need to verify the chain directly; Coda is doing this already, albeit with a simplified blockchain design that is heavily optimized for provability. Such proofs can significantly assist in improving the blockchain's safety and scalability. That said, the total amount of computation that realistically needs to be done is still much less than the amount that's currently done by proof of work miners, so this would at best be an add-on for proof of stake blockchains, not a full-on consensus algorithm.

8. Proof of Stake

Another approach to solving the mining centralization problem is to abolish mining entirely, and move to some other mechanism for counting the weight of each node in the consensus. The most popular alternative under discussion to date is "proof of stake" - that is to say, instead of treating the consensus model as "one unit of CPU power, one vote" it becomes "one currency unit, one vote".

Status: **Great theoretical progress, pending more real-world evaluation**.

Near the end of 2014, it became clear to the proof of stake community that some form of "weak subjectivity" is unavoidable. To maintain economic security, nodes need to obtain a recent checkpoint extra-protocol when they sync for the first time, and again if they go offline for more than a few months. This was a difficult pill to swallow; many PoW advocates still cling to PoW precisely because in a PoW chain the "head" of the chain can be discovered with the only data coming from a trusted source being the blockchain client software itself. PoS advocates, however, were willing to swallow the pill, seeing the added trust requirements as not being large. From there the path to proof of stake through long-duration security deposits became clear.

Most interesting consensus algorithms today are fundamentally similar to PBFT, but replace the fixed set of validators with a dynamic list that anyone can join by sending tokens into a system-level smart contract with time-locked withdrawals (eg. a withdrawal might in some cases take up to 4 months to complete). In many cases (including ethereum 2.0), these algorithms achieve "economic finality" by penalizing validators that are caught performing actions that violate the protocol in certain ways (see here for a philosophical view on what proof of stake accomplishes).

As of today, we have (among many other algorithms):

- **Casper FFG**: https://arxiv.org/abs/1710.09437
- **Tendermint**: https://tendermint.com/docs/spec/consensus/consensus.html
- **HotStuff**: https://arxiv.org/abs/1803.05069
- **Casper CBC**: https://vitalik.ca/general/2018/12/05/cbc_casper.html

There continues to be ongoing refinement (eg. here and here) . Eth2 phase 0, the chain that will implement FFG, is currently under implementation and enormous progress has been made. Additionally, Tendermint has been running, in the form of the Cosmos chain for several months. Remaining arguments about proof of stake, in my view, have to do with optimizing the economic incentives, and further formalizing the strategy for responding to 51% attacks. Additionally, the Casper CBC spec could still use concrete efficiency improvements.

9. Proof of Storage

A third approach to the problem is to use a scarce computational resource other than computational power or currency. In this regard, the two main alternatives that have been proposed are storage and bandwidth. There is no way in principle to provide an after-the-fact cryptographic proof that bandwidth was given or used, so proof of bandwidth should most accurately be considered a subset of social proof, discussed in later problems, but

proof of storage is something that certainly can be done computationally. An advantage of proof-of-storage is that it is completely ASIC-resistant; the kind of storage that we have in hard drives is already close to optimal.

Status: **A lot of theoretical progress, though still a lot to go, as well as more real-world evaluation**.

There are a number of [blockchains planning to use proof of storage](#) protocols, including [Chia](#) and [Filecoin](#). That said, these algorithms have not been tested in the wild. My own main concern is centralization: will these algorithms actually be dominated by smaller users using spare storage capacity, or will they be dominated by large mining farms?

Economics

10. Stable-value cryptoassets

One of the main problems with Bitcoin is the issue of price volatility ... Problem: construct a cryptographic asset with a stable price.

Status: **Some progress**.

[MakerDAO](#) is now live, and has been holding stable for nearly two years. It has survived a 93% drop in the value of its underlying collateral asset (ETH), and there is now more than $100 million in DAI issued. It has become a mainstay of the Ethereum ecosystem, and many Ethereum projects have or are integrating with it. Other synthetic token projects, such as [UMA](#), are rapidly gaining steam as well.

However, while the MakerDAO system has survived tough economic conditions in 2019, the conditions were by no means the toughest that could happen. In the past, Bitcoin has [fallen by 75%](#) over the course of two days; the same may happen to ether or any other collateral asset some day. Attacks on the underlying blockchain are an even larger untested risk, especially if compounded by price decreases at the same time. Another major challenge, and arguably the larger one, is that the stability of MakerDAO-like systems is dependent on some underlying oracle scheme. Different attempts at oracle systems do exist (see #16), but the jury is still out on how well they can hold up under large amounts of economic stress. So far, the collateral controlled by MakerDAO has been lower than the value of the MKR token; if this relationship reverses MKR holders may have a collective incentive to try to "loot" the MakerDAO system. There are ways to try to protect against such attacks, but they have not been tested in real life.

11. Decentralized Public Goods Incentivization

One of the challenges in economic systems in general is the problem of "public goods". For example, suppose that there is a scientific research project which will cost $1 million to complete, and it is known that if it is completed the resulting research will save one million people $5 each. In total, the social benefit is clear ... [but] from the point of view of each individual person contributing does not make sense ... So far, most problems to public goods have involved centralization Additional Assumptions And Requirements: A fully trustworthy oracle exists for determining whether or not a certain public good task has been completed (in reality this is false, but this is the domain of another problem)

Status: **Some progress**.

The problem of funding public goods is generally understood to be split into two problems: the funding problem (where to get funding for public goods from) and the preference aggregation

problem (how to determine what is a genuine public good, rather than some single individual's pet project, in the first place). This problem focuses specifically on the former, assuming the latter is solved (see the "decentralized contribution metrics" section below for work on that problem).

In general, there haven't been large new breakthroughs here. There's two major categories of solutions. First, we can try to elicit individual contributions, giving people social rewards for doing so. My own proposal for charity through marginal price discrimination is one example of this; another is the anti-malaria donation badges on Peepeth. Second, we can collect funds from applications that have network effects. Within blockchain land there are several options for doing this:

- Issuing coins
- Taking a portion of transaction fees at protocol level (eg. through EIP 1559)
- Taking a portion of transaction fees from some layer-2 application (eg. Uniswap, or some scaling solution, or even state rent in an execution environment in ethereum 2.0)
- Taking a portion of other kinds of fees (eg. ENS registration)

Outside of blockchain land, this is just the age-old question of how to collect taxes if you're a government, and charge fees if you're a business or other organization.

12. Reputation systems

Problem: design a formalized reputation system, including a score rep(A,B) -> V where V is the reputation of B from the point of view of A, a mechanism for determining the probability that one party can be trusted by another, and a mechanism for updating the reputation given a record of a particular open or finalized interaction.

Status: **Slow progress**.

There hasn't really been much work on reputation systems since 2014. Perhaps the best is the use of token curated registries to create curated lists of trustable entities/objects; the Kleros ERC20 TCR (yes, that's a token-curated registry of legitimate ERC20 tokens) is one example, and there is even an alternative interface to Uniswap (http://uniswap.ninja) that uses it as the backend to get the list of tokens and ticker symbols and logos from. Reputation systems of the subjective variety have not really been tried, perhaps because there is just not enough information about the "social graph" of people's connections to each other that has already been published to chain in some form. If such information starts to exist for other reasons, then subjective reputation systems may become more popular.

13. Proof of excellence

One interesting, and largely unexplored, solution to the problem of [token] distribution specifically (there are reasons why it cannot be so easily used for mining) is using tasks that are socially useful but require original human-driven creative effort and talent. For example, one can come up with a "proof of proof" currency that rewards players for coming up with mathematical proofs of certain theorems

Status: **No progress, problem is largely forgotten**.

The main alternative approach to token distribution that has instead become popular is airdrops; typically, tokens are distributed at launch either proportionately to existing holdings of some other token, or based on some other metric (eg. as in the Handshake airdrop). Verifying human creativity directly has not really been attempted, and with recent progress on AI the problem of creating a task that only humans can do but computers can verify may well be too difficult.

15 [sic]. Anti-Sybil systems

A problem that is somewhat related to the issue of a reputation system is the challenge of creating a "unique identity system" - a system for generating tokens that prove that an identity is not part of a Sybil attack ... However, we would like to have a system that has nicer and more egalitarian features than "one-dollar-one-vote"; arguably, one-person-one-vote would be ideal.

Status: **Some progress**.

There have been quite a few attempts at solving the unique-human problem. Attempts that come to mind include (incomplete list!):

- **HumanityDAO**: https://www.humanitydao.org/
- **Pseudonym parties**: https://bford.info/pub/net/sybil.pdf
- **POAP** ("proof of attendance protocol"): https://www.poap.xyz/
- **BrightID**: https://www.brightid.org/

With the growing interest in techniques like quadratic voting and quadratic funding, the need for some kind of human-based anti-sybil system continues to grow. Hopefully, ongoing development of these techniques and new ones can come to meet it.

14 [sic]. Decentralized contribution metrics

Incentivizing the production of public goods is, unfortunately, not the only problem that centralization solves. The other problem is determining, first, which public goods are worth producing in the first place and, second, determining to what extent a particular effort actually accomplished the production of the public good. This challenge deals with the latter issue.

Status: **Some progress, some change in focus**.

More recent work on determining value of public-good contributions does not try to separate determining tasks and determining quality of completion; the reason is that in practice the two are difficult to separate. Work done by specific teams tends to be non-fungible and subjective enough that the most reasonable approach is to look at relevance of task and quality of performance as a single package, and use the same technique to evaluate both.

Fortunately, there has been great progress on this, particularly with the discovery of quadratic funding. Quadratic funding is a mechanism where individuals can make donations to projects, and then based on the number of people who donated and how much they donated, a formula is used to calculate how much they would have donated if they were perfectly coordinated with each other (ie. took each other's interests into account and did not fall prey to the tragedy of the commons). The difference between amount would-have-donated and amount actually donated for any given project is given to that project as a subsidy from some central pool (see #11 for where the central pool funding could come from). Note that this mechanism focuses on satisfying the values of some community, not on satisfying some given goal regardless of whether or not anyone cares about it. Because of the complexity of values problem, this approach is likely to be much more robust to unknown unknowns.

Quadratic funding has even been tried in real life with considerable success in the recent gitcoin quadratic funding round. There has also been some incremental progress on improving quadratic funding and similar mechanisms; particularly, pairwise-bounded quadratic funding to mitigate collusion. There has also been work on specification and implementation of bribe-resistant voting technology, preventing users from proving to third parties who they voted for; this prevents many kinds of collusion and bribe attacks.

16. Decentralized success metrics

Problem: come up with and implement a decentralized method for measuring numerical real-world variables ... the system should be able to measure anything that humans can currently reach a rough consensus on (eg. price of an asset, temperature, global CO2 concentration)

Status: **Some progress**.

This is now generally just called "the oracle problem". The largest known instance of a decentralized oracle running is Augur, which has processed outcomes for millions of dollars of bets. Token curated registries such as the Kleros TCR for tokens are another example. However, these systems still have not seen a real-world test of the forking mechanism (search for "subjectivocracy" here) either due to a highly controversial question or due to an attempted 51% attack. There is also research on the oracle problem happening outside of the blockchain space in the form of the "peer prediction" literature; see here for a very recent advancement in the space.

Another looming challenge is that people want to rely on these systems to guide transfers of quantities of assets larger than the economic value of the system's native token. In these conditions, token holders in theory have the incentive to collude to give wrong answers to steal the funds. In such a case, the system would fork and the original system token would likely become valueless, but the original system token holders would still get away with the returns from whatever asset transfer they misdirected. Stablecoins (see #10) are a particularly egregious case of this. One approach to solving this would be a system that assumes that altruistically honest data providers do exist, and creating a mechanism to identify them, and only allowing them to churn slowly so that if malicious ones start getting voted in the users of systems that rely on the oracle can first complete an orderly exit. In any case, more development of oracle tech is very much an important problem.

New problems

If I were to write the hard problems list again in 2019, some would be a continuation of the above problems, but there would be significant changes in emphasis, as well as significant new problems. Here are a few picks:

- **Cryptographic obfuscation**: same as #4 above
- **Ongoing work on post-quantum cryptography**: both hash-based as well as based on post-quantum-secure "structured" mathematical objects, eg. elliptic curve isogenies, lattices...
- **Anti-collusion infrastructure**: ongoing work and refinement of https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413, including adding privacy against the operator, adding multi-party computation in a maximally practical way, etc.
- **Oracles**: same as #16 above, but removing the emphasis on "success metrics" and focusing on the general "get real-world data" problem
- **Unique-human identities** (or, more realistically, semi-unique-human identities): same as what was written as #15 above, but with an emphasis on a less "absolute" solution: it should be much harder to get two identities than one, but making it impossible to get multiple identities is both impossible and potentially harmful even if we do succeed
- **Homomorphic encryption and multi-party computation**: ongoing improvements are still required for practicality
- **Decentralized governance mechanisms**: DAOs are cool, but current DAOs are still very primitive; we can do better
- **Fully formalizing responses to PoS 51% attacks**: ongoing work and refinement of https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363
- **More sources of public goods funding**: the ideal is to charge for congestible resources inside of systems that have network effects (eg. transaction fees), but doing so in decentralized systems requires public legitimacy; hence this is a social problem along with the technical one of finding possible sources
- **Reputation systems**: same as #12 above

In general, base-layer problems are slowly but surely decreasing, but application-layer problems are only just getting started.

Review of Gitcoin Quadratic Funding Round 3

2019 Oct 24

Special thanks to the Gitcoin team and especially Frank Chen for working with me through these numbers

The next round of Gitcoin Grants quadratic funding has just finished, and we the numbers for how much each project has received [were just released](). Here are the top ten:
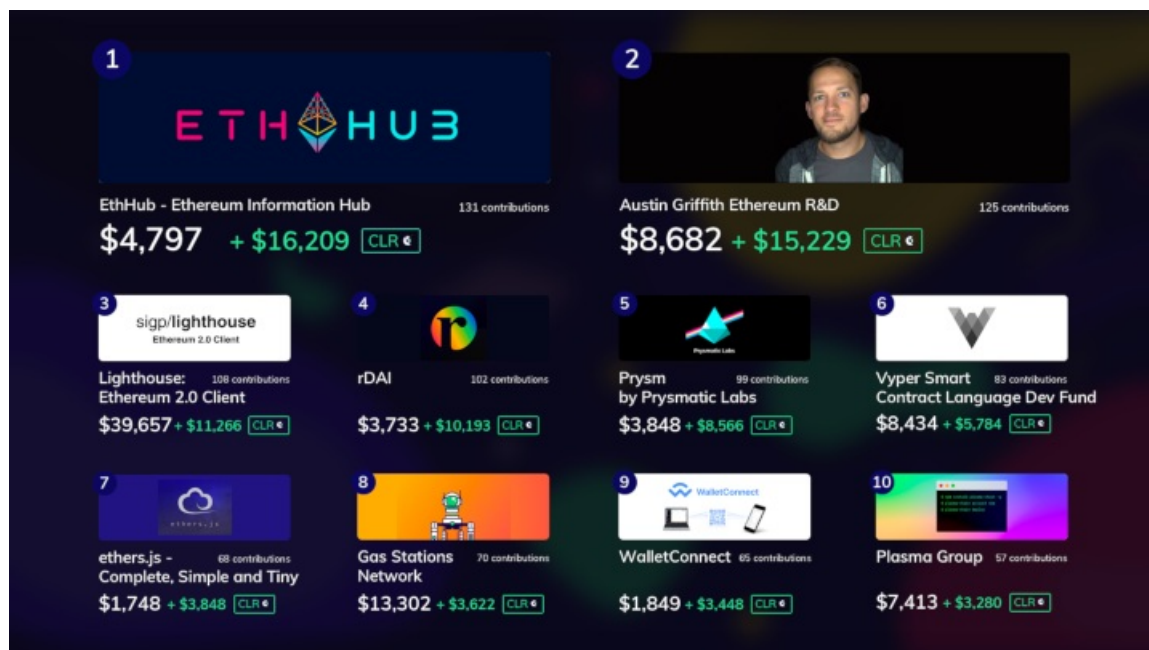
Altogether, $163,279 was donated to 80 projects by 477 contributors, augmented by a matching pool of $100,000. Nearly half came from four contributions above $10,000: $37,500 to Lighthouse, and $12,500 each to Gas Station Network, Black Girls Code and Public Health Incentives Layer. Out of the remainder, about half came from contributions between $1,000 and $10,000, and the rest came from smaller donations of various sizes. But what matters more here are not the raw donations, but rather the subsidies that the [quadratic funding]() mechanism applied. Gitcoin Grants is there to support valuable public goods in the Ethereum ecosystem, but also serve as a testbed for this new quadratic donation matching mechanism, and see how well it lives up to its promise of creating a democratic, market-based and efficient way of funding public goods. This time around, a modified formula based on [pairwise-bounded coordination subsidies]() was used, which has the goal of minimizing distortion from large contributions from coordinated actors. And now we get to see how the experiment went.

Judging the Outcomes

First, the results. Ultimately, every mechanism for allocating resources, whether centralized, market-based, democratic or otherwise, must stand the test of delivering results, or else sooner or later it will be abandoned for another mechanism that is perceived to be better, even if it is less philosophically clean. Judging results is inherently a subjective exercise; any single person's analysis of a mechanism will inevitably be shaped by how well the results fit their own preferences and tastes. However, in those cases where a mechanism does output a surprising result, one can and should use that as an opportunity to learn, and see whether or not one missed some key information that other participants in the mechanism had.

In my own case, I found the top results very agreeable and a quite reasonable catalogue of projects that are good for the Ethereum community. One of the disparities between these grants and the Ethereum Foundation grants is that the Ethereum Foundation grants (see recent rounds [here]() and

[here](#)) tend to overwhelmingly focus on technology with only a small section on education and community resources, whereas in the Gitcoin grants while technology still dominates, EthHub is #2 and lower down defiprime.com is #14 and cryptoeconomics.study is #17. In this case my personal opinion is that EF *has* made a genuine error in undervaluing grants to community/education organizations and Gitcoin's "collective instinct" is correct. Score one for new-age fancy quadratic market democracy.

Another surprising result to me was Austin Griffith getting second place. I personally have never spent too much time thinking about Burner Wallet; I knew that it existed but in my mental space I did not take it too seriously, focusing instead on client development, L2 scaling, privacy and to a lesser extent smart contract wallets (the latter being a key use case of Gas Station Network at #8). After seeing Austin's impressive performance in this Gitcoin round, I asked a few people what was going on.

Burner Wallet ([website](#), [explainer article](#)) is an "insta-wallet" that's very easy to use: just load it up on your desktop or phone, and there you have it. It was used successfully at EthDenver to sell food from food trucks, and generally many people appreciate its convenience. Its main weaknesses are lower security and that one of its features, support for xDAI, is dependent on a permissioned chain.

Austin's Gitcoin grant is there to fund his [ongoing work](#), and I have heard one criticism: there's many prototypes, but comparatively few "things taken to completion". There is also the critique that as great as Austin is, it's difficult to argue that he's as important to the success of Ethereum as, say, Lighthouse and Prysmatic, though one can reply that what matters is not total value, but rather the marginal value of giving a given project or person an extra $10,000. On the whole, however, I feel like quadratic funding's (Glen would say deliberate!) tendency to select for things like Burner Wallet with populist appeal is a much needed corrective to the influence of the Ethereum tech elite (including myself!) who often value technical impressiveness and undervalue simple and quick things that make it really easy for people to participate in Ethereum. This one is slightly more ambiguous, but I'll say score two for new-age fancy quadratic market democracy.

The main thing that I was disappointed the Gitcoiner-ati did *not* support more was Gitcoin maintenance itself. The Gitcoin Sustainability Fund only got a total $1,119 in raw contributions from 18 participants, plus a match of $202. The optional 5% tips that users could give to Gitcoin upon donating were not included into the quadratic matching calculations, but raised another ~$1,000. Given the amount of effort the Gitcoin people put in to making quadratic funding possible, this is not nearly enough; Gitcoin clearly deserves more than 0.9% of the total donations in the round. Meanwhile, the Ethereum Foundation (as well as Consensys and individual donors) have been giving grants to Gitcoin that include supporting Gitcoin itself. Hopefully in future rounds people will support Gitcoin itself too, but for now, score one for good old-fashioned EF technocracy.

On the whole, quadratic funding, while still young and immature, seems to be a remarkably effective complement to the funding preferences of existing institutions, and it seems worthwhile to continue it and even increase its scope and size in the future.

Pairwise-bounded quadratic funding vs traditional quadratic funding

Round 3 differs from previous rounds in that it uses a new [flavor of quadratic funding](#), which limits the subsidy per pair of participants. For example, in traditional QF, if two people each donate $10, the subsidy would be $10, and if two people each donate $10,000, the subsidy would be $10,000. This property of traditional QF makes it highly vulnerable to collusion: two key employees of a project (or even two fake accounts owned by the same person) could each donate as much money as they have, and get back a very large subsidy. Pairwise-bounded QF computes the total subsidy to a project by looking through all pairs of contributors, and imposes a maximum bound on the total subsidy that any given pair of participants can trigger (combined across all projects). Pairwise-bounded QF also has the property that it generally penalizes projects that are dominated by large contributors:
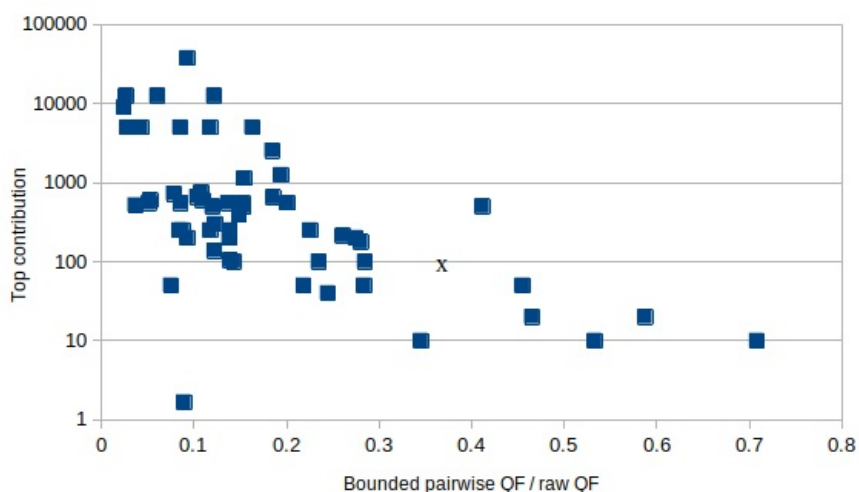
The projects that lost the most relative to traditional QF seem to be projects that have a single large contribution (or sometimes two). For example, "fuzz geth and Parity for EVM consensus bugs" got a $415 match compared to the $2000 he would have gotten in traditional QF; the decrease is explained by the fact that the contributions are dominated by two large $4500 contributions. On the other hand, cryptoeconomics.study got $1274, *up* nearly double from the $750 it would have gotten in traditional QF; this is explained by the large diversity of contributions that the project received and particularly the lack of large sponsors: the largest contribution to cryptoeconomics.study was $100.

Another desirable property of pairwise-bounded QF is that it privileges *cross-tribal* projects. That is, if there are projects that group A typically supports, and projects that group B typically supports, then projects that manage to get support from both groups get a more favorable subsidy (because the pairs that go between groups are not as saturated). Has this incentive for building bridges appeared in these results?

Unfortunately, my code of honor as a social scientist obliges me to report the negative result: the Ethereum community just does not yet have enough internal tribal structure for effects like this to materialize, and even when there are differences in correlations they don't seem strongly connected to higher subsidies due to pairwise-bounding. Here are the cross-correlations between who contributed to different projects:
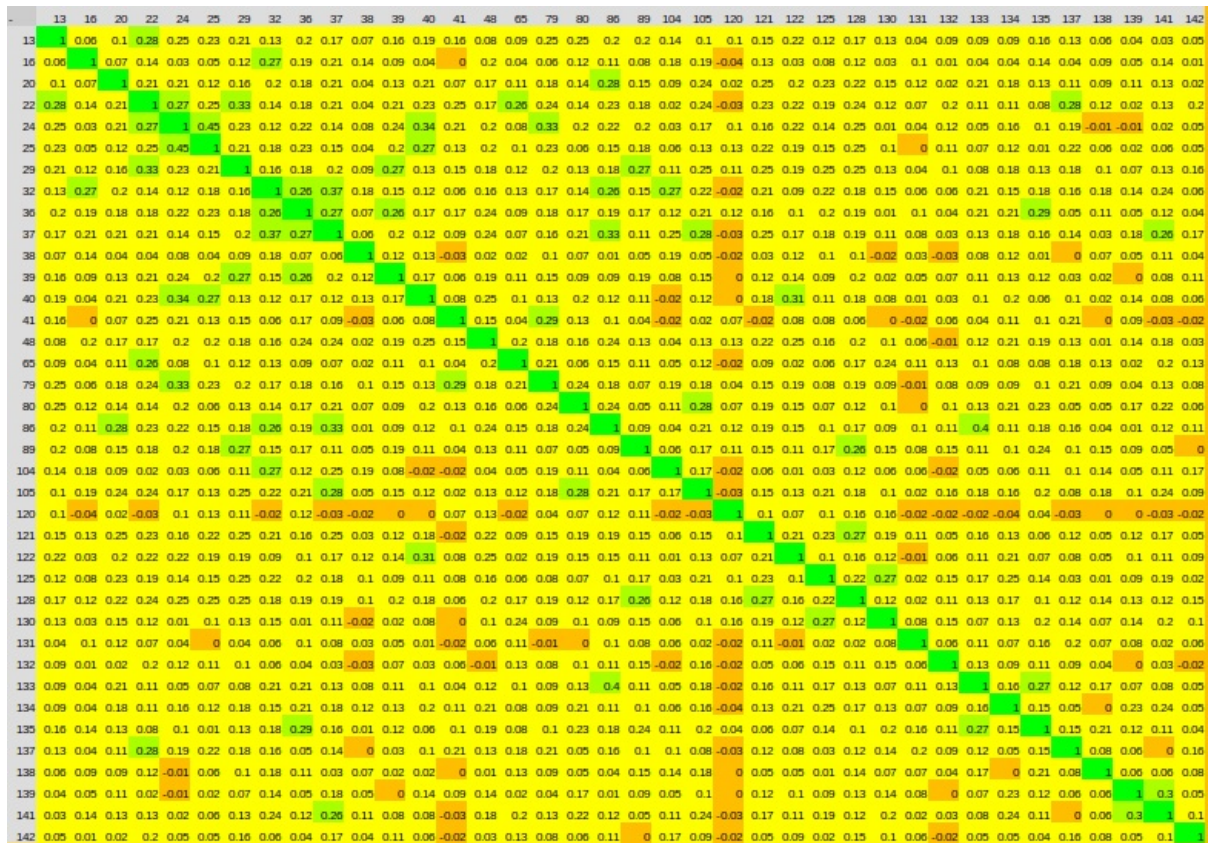
Generally, all projects are slightly positively correlated with each other, with a few exceptions with greater correlation and one exception with broad roughly zero correlation: Nori (120 in this chart). However, Nori did not do well in pairwise-bounded QF, because over 94% of its donations came from a single $5000 donation.

Dominance of large projects

One other pattern that we saw in this round is that popular projects got disproportionately large grants:
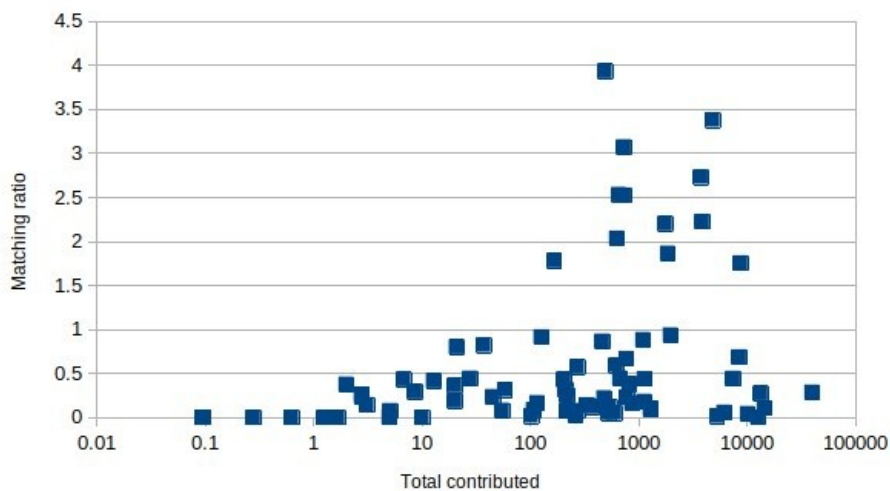
To be clear, this is not just saying "more contributions, more match", it's saying "more contributions, *more match per dollar contributed*". Arguably, this is an intended feature of the mechanism. Projects

that can get more people to donate to them represent public goods that serve a larger public, and so tragedy of the commons problems are more severe and hence contributions to them should be multiplied more to compensate. However, looking at the list, it's hard to argue that, say, Prysm ($3,848 contributed, $8,566 matched) is a more public good than Nimbus ($1,129 contributed, $496 matched; for the unaware, Prysm and Nimbus are both eth2 clients). The failure does not look too severe; on average, projects near the top do seem to serve a larger public and projects near the bottom do seem niche, but it seems clear that at least part of the disparity is not genuine publicness of the good, but rather inequality of attention. N units of marketing effort can attract attention of N people, and theoretically get N^2 resources.

Of course, this could be solved via a "layer on top" venture-capital style: upstart new projects could get investors to support them, in return for a share of matched contributions received when they get large. Something like this would be needed eventually; predicting future public goods is as important a social function as predicting future private goods. But we could also consider less winner-take-all alternatives; the simplest one would be adjusting the QF formula so it uses an exponent of eg. 1.5 instead of 2. I can see it being worthwhile to try a future round of Gitcoin Grants with such a formula ($\left(\sum_i x_i^{\frac{2}{3}}\right)^{\frac{3}{2}}$ instead of $\left(\sum_i x_i^{\frac{1}{2}}\right)^2$) to see what the results are like.

Individual leverage curves

One key question is, if you donate $1, or $5, or $100, how big an impact can you have on the amount of money that a project gets? Fortunately, we can use the data to calculate these deltas!
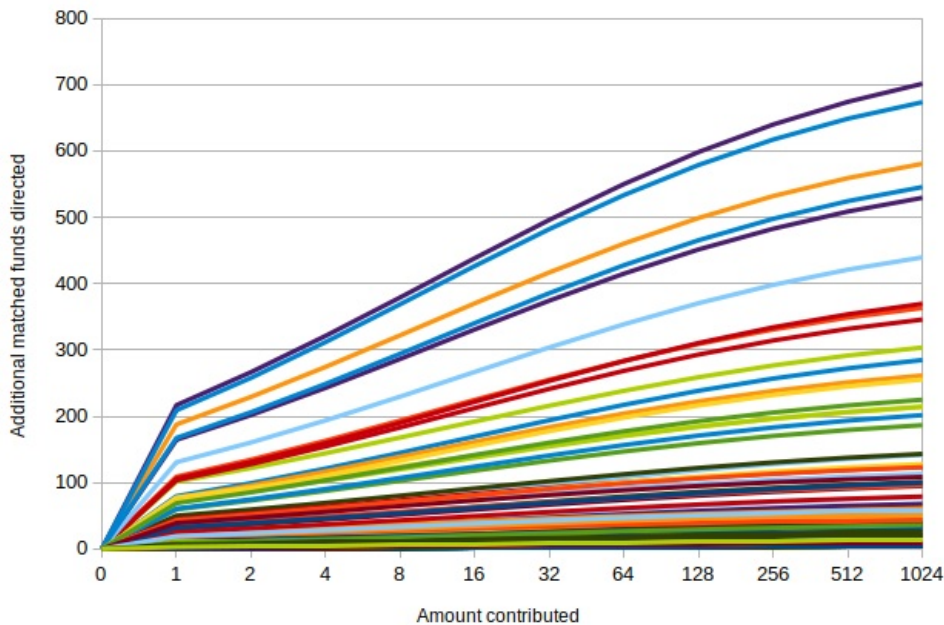
The different lines are for different projects; supporting projects with higher existing support will lead to you getting a bigger multiplier. In all cases, the first dollar is very valuable, with a matching ratio in some cases over 100:1. But the second dollar is much less valuable, and matching ratios quickly taper off; even for the largest projects increasing one's donation from $32 to $64 will only get a 1:1 match, and anything above $100 becomes almost a straight donation with nearly no matching. However, given that it's likely possible to get legitimate-looking Github accounts on the grey market for around those costs, having a cap of a few hundred dollars on the amount of matched funds that any particular account can direct seems like a very reasonable mitigation, despite its costs in limiting the bulk of the matching effect to small-sized donations.

Conclusions

On the whole, this was by far the largest and the most data-rich Gitcoin funding round to date. It successfully attracted hundreds of contributors, reaching a size where we can finally see many significant effects in play and drown out the effects of the more naive forms of small-scale collusion. The experiment already seems to be leading to valuable information that can be used by future

quadratic funding implementers to improve their quadratic funding implementations. The case of Austin Griffith is also interesting because $23,911 in funds that he received comes, in relative terms, surprisingly close to an average salary for a developer if the grants can be repeated on a regular schedule. What this means is that if Gitcoin Grants *does* continue operating regularly, and attracts and expands its pool of donations, we could be very close to seeing the first "quadratic freelancer" - someone directly "working for the public", funded by donations boosted by quadratic matching subsidies. And at that point we could start to see more experimentation in new forms of organization that live on top of quadratic funding gadgets as a base layer. All in all, this foretells an exciting and, err, radical public-goods funding future ahead of us.

In-person meatspace protocol to prove unconditional possession of a private key

2019 Oct 01

Recommended pre-reading: https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413

Alice slowly walks down the old, dusty stairs of the building into the basement. She thinks wistfully of the old days, when quadratic-voting in the World Collective Market was a much simpler process of linking her public key to a twitter account and opening up metamask to start firing off votes. Of course back then voting in the WCM was used for little; there were a few internet forums that used it for voting on posts, and a few million dollars donated to its [quadratic funding](#) oracle. But then it grew, and then the game-theoretic attacks came.

First came the exchange platforms, which started offering "[dividends](#)" to anyone who registered a public key belonging to an exchange and thus provably allowed the exchange to vote on their behalf, breaking the crucial "independent choice" assumption of the quadratic voting and funding mechanisms. And soon after that came the fake accounts - Twitter accounts, Reddit accounts filtered by karma score, national government IDs, all proved vulnerable to either government cheating or hackers, or both. Elaborate infrastructure was instituted at registration time to ensure both that account holders were real people, and that account holders themselves held the keys, not a central custody service purchasing keys by the thousands to buy votes.

And so today, voting is still easy, but initiation, while still not harder than going to a government office, is no longer exactly trivial. But of course, with billions of dollars in donations from now-deceased billionaires and cryptocurrency premines forming part of the WCM's quadratic funding pool, and elements of municipal governance using its quadratic voting protocols, participating is very much worth it.

After reaching the end of the stairs, Alice opens the door and enters the room. Inside the room, she sees a table. On the near side of the table, she sees a single, empty chair. On the far side of the table, she sees four people already sitting down on chairs of their own, the high-reputation Guardians randomly selected by the WCM for Alice's registration ceremony. "Hello, Alice," the person sitting on the leftmost chair, whose name she intuits is Bob, says in a calm voice. "Glad that you can make it," the person sitting beside Bob, whose name she intuits is Charlie, adds.

Alice walks over to the chair that is clearly meant for her and sits down. "Let us begin," the person sitting beside Charlie, whose name by logical progression is David, proclaims. "Alice, do you have your key shares?"

Alice takes out four pocket-sized notebooks, clearly bought from a dollar store, and places them on the table. The person sitting at the right, logically named Evan, takes out his phone, and immediately the others take out theirs. They open up their ethereum wallets. "So," Evan begins, "the current Ethereum beacon chain slot number is 28,205,913, and the block hash starts `0xbe48`. Do all agree?". "Yes," Alice, Bob, Charlie and David exclaim in unison. Evan continues: "so let us wait for the next block."

The five intently stare at their phones. First for ten seconds, then twenty, then thirty. "Three skipped proposers," Bob mutters, "how unusual". But then after another ten seconds, a new block appears. "Slot number 28,205,917, block hash starts `0x62f9`, so first digit 6. All agreed?"

"Yes."

"Six mod four is two, and as is prescribed in the Old Ways, we start counting indices from zero, so this means Alice will keep the third book, counting as usual from our left."

Bob takes the first, second and fourth notebooks that Alice provided, leaving the third untouched. Alice takes the remaining notebook and puts it back in her backpack. Bob opens each notebook to a page in the middle with the corner folded, and sees a sequence of letters and numbers written with a pencil in the middle of each page - a standard way of writing the key shares for over a decade, since camera and image processing technology got powerful enough to recognize words and numbers written on single slips of paper even inside an envelope. Bob, Charlie, David and Evan crowd around

the books together, and each open up an app on their phone and press a few buttons.

Bob starts reading, as all four start typing into their phones at the same time:

"Alice's first key share is, `6-b-d-7-h-k-k-l-o-e-q-q-p-3-y-s-6-x-e-f`. Applying the 100,000x iterated SHA256 hash we get `e-a-6-6...`, confirm?"

"Confirmed," the others replied. "Checking against Alice's precommitted elliptic curve point A0... match."

"Alice's second key share is, `f-r-n-m-j-t-x-r-s-3-b-u-n-n-n-i-z-3-d-g`. Iterated hash `8-0-3-c...`, confirm?"

"Confirmed. Checking against Alice's precommitted elliptic curve point A1... match."

"Alice's fourth key share is, `i-o-f-s-a-q-f-n-w-f-6-c-e-a-m-s-6-z-z-n`. Iterated hash `6-a-5-6...`, confirm?"

"Confirmed. Checking against Alice's precommitted elliptic curve point A3... match."

"Adding the four precommitted curve points, x coordinate begins `3-1-8-3`. Alice, confirm that that is the key you wish to register?"

"Confirm."

Bob, Charlie, David and Evan glance down at their smartphone apps one more time, and each tap a few buttons. Alice catches a glance at Charlie's phone; she sees four yellow checkmarks, and an "approval transaction pending" dialog. After a few seconds, the four yellow checkmarks are replaced with a single green checkmark, with a transaction hash ID, too small for Alice to make out the digits from a few meters away, below. Alice's phone soon buzzes, with a notification dialog saying "Registration confirmed".

"Congratulations, Alice," Bob says. "Unconditional possession of your key has been verified. You are now free to send a transaction to the World Collective Market's MPC oracle to update your key."

"Only a 75% probability this would have actually caught me if I didn't actually have all four parts of the key," Alice thought to herself. But it seemed to be enough for an in-person protocol in practice; and if it ever wasn't then they could easily switch to slightly more complex protocols that used low-degree polynomials to achieve exponentially high levels of soundness. Alice taps a few buttons on her smartphone, and a "transaction pending" dialog shows up on the screen. Five seconds later, the dialog disappears and is replaced by a green checkmark. She jumps up with joy and, before Bob, Charlie, David and Evan can say goodbye, runs out of the room, frantically tapping buttons to vote on all the projects and issues in the WCM that she had wanted to support for months.

Understanding PLONK

2019 Sep 22

Special thanks to Justin Drake, Karl Floersch, Hsiao-wei Wang, Barry Whitehat, Dankrad Feist, Kobi Gurkan and Zac Williamson for review

Very recently, Ariel Gabizon, Zac Williamson and Oana Ciobotaru announced a new general-purpose zero-knowledge proof scheme called PLONK, standing for the unwieldy quasi-backronym "Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge". While improvements to general-purpose zero-knowledge proof protocols have been coming for years, what PLONK (and the earlier but more complex SONIC and the more recent Marlin) bring to the table is a series of enhancements that may greatly improve the usability and progress of these kinds of proofs in general.

The first improvement is that while PLONK still requires a trusted setup procedure similar to that needed for the SNARKs in Zcash, it is a "universal and updateable" trusted setup. This means two things: first, instead of there being one separate trusted setup for every program you want to prove things about, there is one single trusted setup for the whole scheme after which you can use the scheme with any program (up to some maximum size chosen when making the setup). Second, there is a way for multiple parties to participate in the trusted setup such that it is secure as long as any one of them is honest, and this multi-party procedure is fully sequential: first one person participates, then the second, then the third... The full set of participants does not even need to be known ahead of time; new participants could just add themselves to the end. This makes it easy for the trusted setup to have a large number of participants, making it quite safe in practice.

The second improvement is that the "fancy cryptography" it relies on is one single standardized component, called a "polynomial commitment". PLONK uses "Kate commitments", based on a trusted setup and elliptic curve pairings, but you can instead swap it out with other schemes, such as FRI (which would turn PLONK into a kind of STARK) or DARK (based on hidden-order groups). This means the scheme is theoretically compatible with any (achievable) tradeoff between proof size and security assumptions.
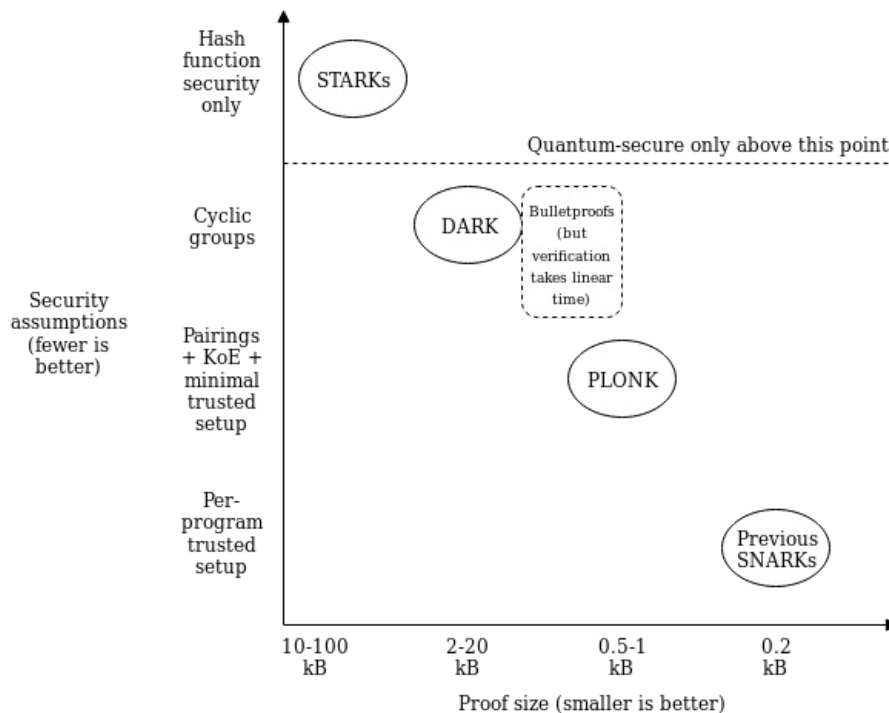
What this means is that use cases that require different tradeoffs between proof size and security assumptions (or developers that have different ideological positions about this question) can still

share the bulk of the same tooling for "arithmetization" - the process for converting a program into a set of polynomial equations that the polynpomial commitments are then used to check. If this kind of scheme becomes widely adopted, we can thus expect rapid progress in improving shared arithmetization techniques.

How PLONK works

Let us start with an explanation of how PLONK works, in a somewhat abstracted format that focuses on polynomial equations without immediately explaining how those equations are verified. A key ingredient in PLONK, as is the case in the [QAPs used in SNARKs](#), is a procedure for converting a problem of the form "give me a value X such that a specific program P that I give you, when evaluated with X as an input, gives some specific result Y" into the problem "give me a set of values that satisfies a set of math equations". The program P can represent many things; for example the problem could be "give me a solution to this sudoku", which you would encode by setting P to be a sudoku verifier plus some initial values encoded and setting Y to 1 (ie. "yes, this solution is correct"), and a satisfying input X would be a valid solution to the sudoku. This is done by representing P as a circuit with logic gates for addition and multiplication, and converting it into a system of equations where the variables are the values on all the wires and there is one equation per gate (eg. $x_6 = x_4 \cdot x_7$ for multiplication, $x_8 = x_5 + x_9$ for addition).

Here is an example of the problem of finding x such that $P(x) = x^3 + x + 5 = 35$ (hint: $x = 3$):
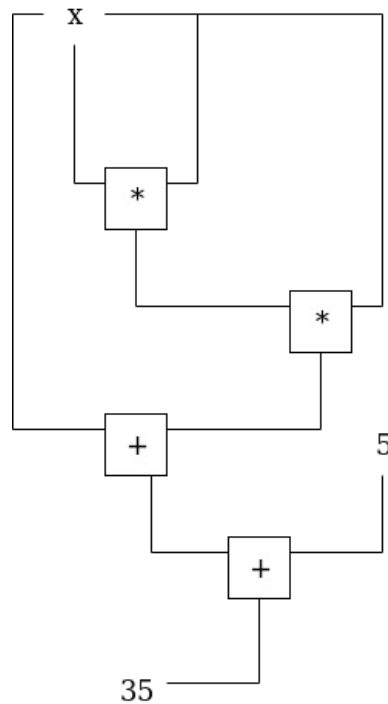
We can label the gates and wires as follows:
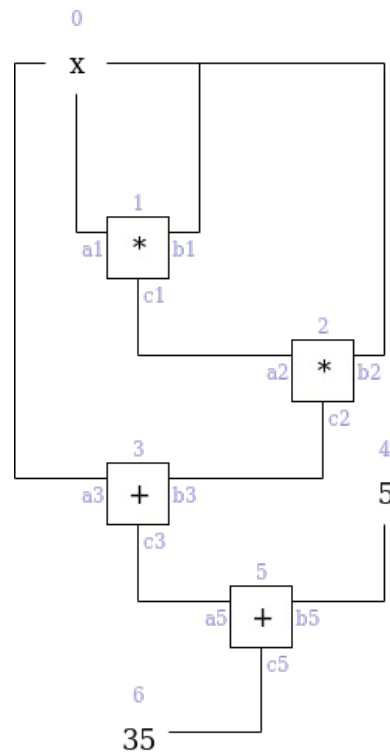
On the gates and wires, we have two types of constraints: **gate constraints** (equations between wires attached to the same gate, eg. $a_1 \cdot b_1 = c_1$) and **copy constraints** (claims about equality of different wires anywhere in the circuit, eg. $a_0 = a_1 = b_1 = b_2 = a_3$) or $(c_0 = a_1)$. We will need to create a structured system of equations, which will ultimately reduce to a very small number of polynomial equations, to represent both.

In PLONK, the setup for these equations is as follows. Each equation is of the following form (think: L = left, R = right, O = output, M = multiplication, C = constant):

$$ \left(Q_{L_{i}}\right) a_{i}+\left(Q_{R_{i}}\right) b_{i}+\left(Q_{O_{i}}\right) c_{i}+\left(Q_{M_{i}}\right) a_{i} b_{i}+Q_{C_{i}}=0 $$

Each Q value is a constant; the constants in each equation (and the number of equations) will be different for each program. Each small-letter value is a variable, provided by the user: a_i is the left input wire of the i'th gate, b_i is the right input wire, and c_i is the output wire of the i'th gate. For an addition gate, we set:

$$ Q_{L_{i}}=1, Q_{R_{i}}=1, Q_{M_{i}}=0, Q_{O_{i}}=-1, Q_{C_{i}}=0 $$

Plugging these constants into the equation and simplifying gives us $a_i + b_i - c_i = 0$, which is exactly the constraint that we want. For a multiplication gate, we set:

$$ Q_{L_{i}}=0, Q_{R_{i}}=0, Q_{M_{i}}=1, Q_{O_{i}}=-1, Q_{C_{i}}=0 $$

For a constant gate setting a_i to some constant x, we set:

$$ Q_{L}=1, Q_{R}=0, Q_{M}=0, Q_{O}=0, Q_{C}=-x $$

You may have noticed that each end of a wire, as well as each wire in a set of wires that clearly must have the same value (eg. x), corresponds to a distinct variable; there's nothing so far forcing the output of one gate to be the same as the input of another gate (what we call "copy constraints"). PLONK does of course have a way of enforcing copy constraints, but we'll get to this later. So now we have a problem where a prover wants to prove that they have a bunch of x_{a_i}, x_{b_i} and (x_{c_i}) values that satisfy a bunch of equations that are of the same form. This is still a big problem, but unlike "find a satisfying input to this computer program" it's a very *structured* big problem, and we have mathematical tools to "compress" it.

From linear systems to polynomials

If you have read about [STARKs](#) or [QAPs](#), the mechanism described in this next section will hopefully feel somewhat familiar, but if you have not that's okay too. The main ingredient here is to understand a *polynomial* as a mathematical tool for encapsulating a whole lot of values into a single object. Typically, we think of polynomials in "coefficient form", that is an expression like:

$$ y=x^{3}-5 x^{2}+7 x-2 $$

But we can also view polynomials in "evaluation form". For example, we can think of the above as being "the" degree < 4 polynomial with evaluations $(-2, 1, 0, 1)$ at the coordinates $(0, 1, 2, 3)$ respectively.
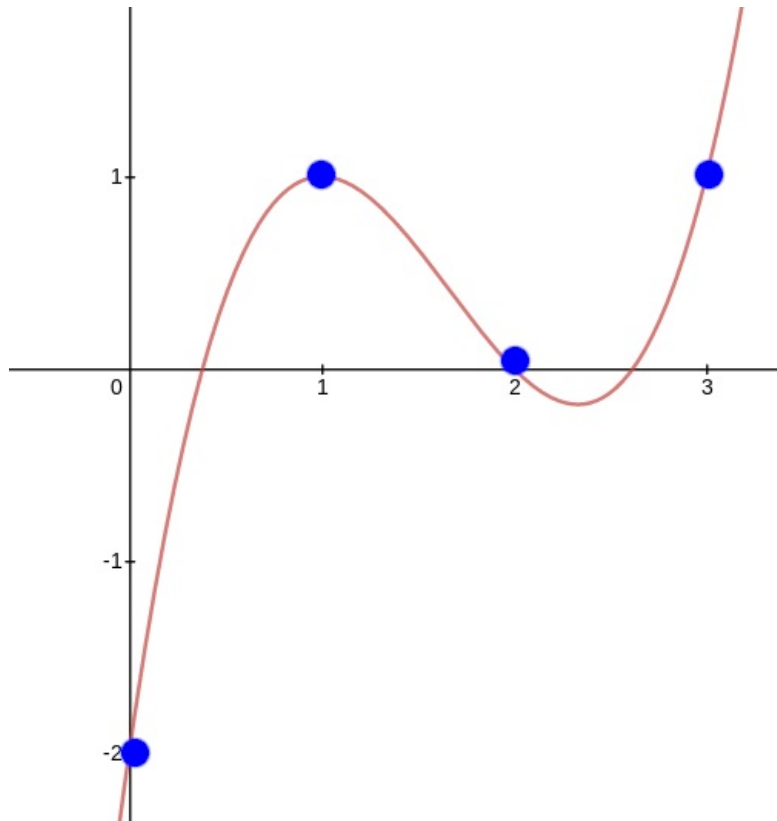
Now here's the next step. Systems of many equations of the same form can be re-interpreted as a single equation over polynomials. For example, suppose that we have the system:

$$ \begin{array}{l}{2 x_{1}-x_{2}+3 x_{3}=8} \\ {x_{1}+4 x_{2}-5 x_{3}=5} \\ {8 x_{1}-x_{2}-x_{3}=-2}\end{array} $$

Let us define four polynomials in evaluation form: $L(x)$ is the degree < 3 polynomial that evaluates to $(2, 1, 8)$ at the coordinates $(0, 1, 2)$, and at those same coordinates $M(x)$ evaluates to $(-1, 4, -1)$, $R(x)$ to $(3, -5, -1)$ and $O(x)$ to $(8, 5, -2)$ (it is okay to directly define polynomials in this way; you can use [Lagrange interpolation](#) to convert to coefficient form). Now, consider the equation:

$$ L(x) \cdot x_{1}+M(x) \cdot x_{2}+R(x) \cdot x_{3}-O(x)=Z(x) H(x) $$

Here, $Z(x)$ is shorthand for $(x-0) \cdot (x-1) \cdot (x-2)$ - the minimal (nontrivial) polynomial that returns zero over the evaluation domain $(0, 1, 2)$. A solution to this equation ($x_1 = 1, x_2 = 6, x_3 = 4, H(x) = 0$) is also a solution to the original system of equations, except the original system does not need $H(x)$. Notice also that in this case, $H(x)$ is conveniently zero, but in more complex cases H may need to be nonzero.

So now we know that we can represent a large set of constraints within a small number of mathematical objects (the polynomials). But in the equations that we made above to represent the gate wire constraints, the x_1, x_2, x_3 variables are different per equation. We can handle this by making the variables themselves polynomials rather than constants in the same way. And so we get:

$$ Q_{L}(x) a(x)+Q_{R}(x) b(x)+Q_{O}(x) c(x)+Q_{M}(x) a(x) b(x)+Q_{C}(x)=0 $$

As before, each Q polynomial is a parameter that can be generated from the program that is being verified, and the a, b, c polynomials are the user-provided inputs.

Copy constraints

Now, let us get back to "connecting" the wires. So far, all we have is a bunch of disjoint equations about disjoint values that are independently easy to satisfy: constant gates can be satisfied by setting the value to the constant and addition and multiplication gates can simply be satisfied by setting all wires to zero! To make the problem actually challenging (and actually represent the problem encoded in the original circuit), we need to add an equation that verifies "copy constraints": constraints such as $a(5) = c(7)$, $c(10) = c(12)$, etc. This requires some clever trickery.

Our strategy will be to design a "coordinate pair accumulator", a polynomial $p(x)$ which works as follows. First, let $X(x)$ and $Y(x)$ be two polynomials representing the x and y coordinates of a set of points (eg. to represent the set $((0, -2), (1, 1), (2, 0), (3, 1))$ you might set $X(x) = x$ and $Y(x) = x^3 - 5x^2 + 7x - 2)$. Our goal will be to let $p(x)$ represent all the points up to (but not including) the given position, so $p(0)$ starts at 1, $p(1)$ represents just the first point, $p(2)$ the first and the second, etc. We will do this by "randomly" selecting two constants, v_1 and v_2, and constructing $p(x)$ using the constraints $p(0) = 1$ and $p(x+1) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$ at least within the domain $((0, 1, 2, 3))$.

For example, letting $v_1 = 3$ and $v_2 = 2$, we get:
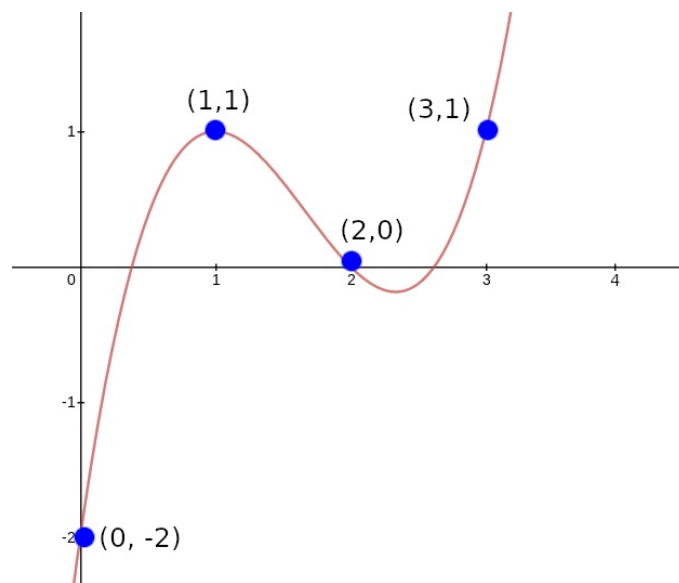
X(x)	0	1	2	3	4
$Y(x)$	-2	1	0	1	
$(v_1 + X(x) + v_2 \cdot Y(x))$	-1	6	5	8	
$p(x)$	1	-1	-6	-30	-240

Notice that (aside from the first column) every $p(x)$ value equals the value to the left of it multiplied by the value to the left and above it.

The result we care about is $p(4) = -240$. Now, consider the case where instead of $X(x) = x$, we set $X(x) = \frac{2}{3} x^3 - 4x^2 + \frac{19}{3}x)$ (that is, the polynomial that evaluates to $((0, 3, 2, 1))$ at the coordinates $((0, 1, 2, 3))$). If you run the same procedure, you'll find that you also get $p(4) = -240$. This is not a coincidence (in fact, if you randomly pick v_1 and v_2 from a sufficiently large field, it will *almost never* happen coincidentally). Rather, this happens because $Y(1) = Y(3)$, so if you "swap the X coordinates" of the points $((1, 1))$ and $((3, 1))$ you're not changing the *set* of points, and because the accumulator encodes a set (as multiplication does not care about order) the value at the end will be the same.

Now we can start to see the basic technique that we will use to prove copy constraints. First, consider the simple case where we only want to prove copy constraints within one set of wires (eg.

we want to prove $a(1) = a(3)$. We'll make two coordinate accumulators: one where $X(x) = x$ and $Y(x) = a(x)$, and the other where $Y(x) = a(x)$ but $X'(x)$ is the polynomial that evaluates to the permutation that flips (or otherwise rearranges) the values in each copy constraint; in the $a(1) = a(3)$ case this would mean the permutation would start $0\ 3\ 2\ 1\ 4...$. The first accumulator would be compressing $((0, a(0)), (1, a(1)), (2, a(2)), (3, a(3)), (4, a(4))...$, the second $((0, a(0)), (3, a(1)), (2, a(2)), (1, a(3)), (4, a(4))...$. The only way the two can give the same result is if $a(1) = a(3)$.

To prove constraints between a, b and c, we use the same procedure, but instead "accumulate" together points from all three polynomials. We assign each of a, b, c a range of X coordinates (eg. a gets $X_a(x) = x$ ie. $(0...n-1)$, b gets $X_b(x) = n+x$, ie. $(n...2n-1)$, c gets $X_c(x) = 2n+x$, ie. $(2n...3n-1)$. To prove copy constraints that hop between different sets of wires, the "alternate" X coordinates would be slices of a permutation across all three sets. For example, if we want to prove $a(2) = b(4)$ with $n = 5$, then $X'_a(x)$ would have evaluations $(\{0, 1, 9, 3, 4\})$ and $X'_b(x)$ would have evaluations $(\{5, 6, 7, 8, 2\})$ (notice the 2 and 9 flipped, where 9 corresponds to the b_4 wire). Often, $X'_a(x)$, $X'_b(x)$ and $X'_c(x)$ are also called $\sigma_a(x)$, $\sigma_b(x)$ and $\sigma_c(x)$.

We would then instead of checking equality within one run of the procedure (ie. checking $p(4) = p'(4)$ as before), we would check *the product* of the three different runs on each side:

$$ p_{a}(n) \cdot p_{b}(n) \cdot p_{c}(n)=p_{a}^{\prime}(n) \cdot p_{b}^{\prime}(n) \cdot p_{c}^{\prime}(n) $$

The product of the three $p(n)$ evaluations on each side accumulates *all* coordinate pairs in the a, b and c runs on each side together, so this allows us to do the same check as before, except that we can now check copy constraints not just between positions within one of the three sets of wires a, b or c, but also between one set of wires and another (eg. as in $a(2) = b(4)$).

And that's all there is to it!

Putting it all together

In reality, all of this math is done not over integers, but over a prime field; check the section "A Modular Math Interlude" here for a description of what prime fields are. Also, for mathematical reasons perhaps best appreciated by reading and understanding this article on FFT implementation, instead of representing wire indices with $x=0....n-1$, we'll use powers of $\omega: 1, \omega, \omega^2....\omega^{n-1}$ where ω is a high-order root-of-unity in the field. This changes nothing about the math, except that the coordinate pair accumulator constraint checking equation changes from $p(x + 1) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$ to $p(\omega \cdot x) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$, and instead of using $(0..n-1)$, $(n..2n-1)$, $(2n..3n-1)$ as coordinates we use $\omega^i, g \cdot \omega^i$ and $g^2 \cdot \omega^i$ where g can be some random high-order element in the field.

Now let's write out all the equations we need to check. First, the main gate-constraint satisfaction check:

$$ Q_{L}(x) a(x)+Q_{R}(x) b(x)+Q_{O}(x) c(x)+Q_{M}(x) a(x) b(x)+Q_{C}(x)=0 $$

Then the polynomial accumulator transition constraint (note: think of "$= Z(x) \cdot H(x)$" as meaning "equals zero for all coordinates within some particular domain that we care about, but not necessarily outside of it"):

$$ \begin{array}{l}{P_{a}(\omega x)-P_{a}(x)\left(v_{1}+x+v_{2} a(x)\right) =Z(x) H_{1}(x)} \\ {P_{a^{\prime}}(\omega x)-P_{a^{\prime}}(x)\left(v_{1}+\sigma_{a}(x)+v_{2} a(x)\right)=Z(x) H_{2}(x)} \\ {P_{b}(\omega x)-P_{b}(x)\left(v_{1}+g x+v_{2} b(x)\right)=Z(x) H_{3}(x)} \\ {P_{b^{\prime}}(\omega x)-P_{b^{\prime}}(x)\left(v_{1}+\sigma_{b}(x)+v_{2} b(x)\right)=Z(x) H_{4}(x)} \\ {P_{c}(\omega x)-P_{c}(x)\left(v_{1}+g^{2} x+v_{2} c(x)\right)=Z(x) H_{5}(x)} \\ {P_{c^{\prime}}(\omega x)-P_{c^{\prime}}(x)\left(v_{1}+\sigma_{c}(x)+v_{2} c(x)\right)=Z(x) H_{6}(x)}\end{array} $$

Then the polynomial accumulator starting and ending constraints:

$$ \begin{array}{l}{P_{a}(1)=P_{b}(1)=P_{c}(1)=P_{a^{\prime}}(1)=P_{b^{\prime}}(1)=P_{c^{\prime}}(1)=1} \\ {P_{a}\left(\omega^{n}\right) P_{b}\left(\omega^{n}\right) P_{c}\left(\omega^{n}\right)=P_{a^{\prime}}\left(\omega^{n}\right) P_{b^{\prime}}\left(\omega^{n}\right) P_{c^{\prime}}\left(\omega^{n}\right)}\end{array} $$

The user-provided polynomials are:

- The wire assignments $a(x), b(x), c(x)$
- The coordinate accumulators $P_a(x), P_b(x), P_c(x), P_{a'}(x), P_{b'}(x), P_{c'}(x)$
- The quotients $H(x)$ and $H_1(x)...H_6(x)$

The program-specific polynomials that the prover and verifier need to compute ahead of time are:

- $Q_L(x), Q_R(x), Q_O(x), Q_M(x), Q_C(x)$, which together represent the gates in the circuit (note that $Q_C(x)$ encodes public inputs, so it may need to be computed or modified at runtime)
- The "permutation polynomials" $\sigma_a(x), \sigma_b(x)$ and $\sigma_c(x)$, which encode the copy constraints between the a, b and c wires

Note that the verifier need only store commitments to these polynomials. The only remaining polynomial in the above equations is $Z(x) = (x - 1) \cdot (x - \omega) \cdot ... \cdot (x - \omega^{n-1})$ which is designed to evaluate to zero at all those points. Fortunately, ω can be chosen to make this polynomial very easy to evaluate: the usual technique is to choose ω to satisfy $\omega^n = 1$, in which case $Z(x) = x^n - 1$.

There is one nuance here: the constraint between $P_a(\omega^{i+1})$ and $P_a(\omega^i)$ can't be true across the *entire* circle of powers of ω; it's almost always false at ω^{n-1} as the next coordinate is $\omega^n = 1$ which brings us back to the *start* of the "accumulator"; to fix this, we can modify the constraint to say "*either* the constraint is true *or* $x = \omega^{n-1}$)", which one can do by multiplying $(x - \omega^{n-1})$ into the constraint so it equals zero at that point.

The only constraint on v_1 and v_2 is that the user must not be able to choose $a(x), b(x)$ or $c(x)$ after v_1 and v_2 become known, so we can satisfy this by computing v_1 and v_2 from hashes of commitments to $a(x), b(x)$ and $c(x)$.

So now we've turned the program satisfaction problem into a simple problem of satisfying a few equations with polynomials, and there are some optimizations in PLONK that allow us to remove many of the polynomials in the above equations that I will not go into to preserve simplicity. But the polynomials themselves, both the program-specific parameters and the user inputs, are **big**. So the next question is, how do we get around this so we can make the proof short?

Polynomial commitments

A [polynomial commitment](#) is a short object that "represents" a polynomial, and allows you to verify evaluations of that polynomial, without needing to actually contain all of the data in the polynomial. That is, if someone gives you a commitment c representing $P(x)$, they can give you a proof that can convince you, for some specific z, what the value of $P(z)$ is. There is a further mathematical result that says that, over a sufficiently big field, if certain kinds of equations (chosen before z is known) about polynomials evaluated at a random z are true, those same equations are true about the whole polynomial as well. For example, if $P(z) \cdot Q(z) + R(z) = S(z) + 5$, then we know that it's overwhelmingly likely that $P(x) \cdot Q(x) + R(x) = S(x) + 5$ in general. Using such polynomial commitments, we could very easily check all of the above polynomial equations above - make the commitments, use them as input to generate z, prove what the evaluations are of each polynomial at z, and then run the equations with these evaluations instead of the original polynomials. But how do these commitments work?

There are two parts: the commitment to the polynomial $P(x) \rightarrow c$, and the opening to a value $P(z)$ at some z. To make a commitment, there are many techniques; one example is [FRI](#), and another is Kate commitments which I will describe below. To prove an opening, it turns out that there is a simple generic "subtract-and-divide" trick: to prove that $P(z) = a$, you prove that

$$ \frac{P(x)-a}{x-z} $$

is also a polynomial (using another polynomial commitment). This works because if the quotient is a polynomial (ie. it is not fractional), then $(x - z)$ is a factor of $(P(x) - a)$, so $(P(x) - a)(z) = 0$, so $P(z) = a$. Try it with some polynomial, eg. $P(x) = x^3 + 2 \cdot x^2 + 5$ with $(z = 6, a = 293)$, yourself; and try $(z = 6, a = 292)$ and see how it fails (if you're lazy, see WolframAlpha [here](#) vs [here](#)). Note also a generic optimization: to prove many openings of many polynomials at the same time, after committing to the outputs do the subtract-and-divide trick on a *random linear combination* of the polynomials and the outputs.

So how do the commitments themselves work? Kate commitments are, fortunately, much simpler than FRI. A trusted-setup procedure generates a set of elliptic curve points $G, G \cdot s, G \cdot$

s^2) .... $(G \cdot s^n)$, as well as $(G_2 \cdot s)$, where G and G_2 are the generators of two elliptic curve groups and s is a secret that is forgotten once the procedure is finished (note that there is a multi-party version of this setup, which is secure as long as at least one of the participants forgets their share of the secret). These points are published and considered to be "the proving key" of the scheme; anyone who needs to make a polynomial commitment will need to use these points. A commitment to a degree-d polynomial is made by multiplying each of the first d+1 points in the proving key by the corresponding coefficient in the polynomial, and adding the results together.

Notice that this provides an "evaluation" of that polynomial at s, without knowing s. For example, $(x^3 + 2x^2+5)$ would be represented by $((G \cdot s^3) + 2 \cdot (G \cdot s^2) + 5 \cdot G)$. We can use the notation $[P]$ to refer to P encoded in this way (ie. $G \cdot P(s)$). When doing the subtract-and-divide trick, you can prove that the two polynomials actually satisfy the relation by using [elliptic curve pairings](#): check that $e([P] - G \cdot a, G_2) = e([Q], [x] - G_2 \cdot z)$ as a proxy for checking that $P(x) - a = Q(x) \cdot (x - z)$.

But there are more recently other types of polynomial commitments coming out too. A new scheme called DARK ("Diophantine arguments of knowledge") uses "hidden order groups" such as [class groups](#) to implement another kind of polynomial commitment. Hidden order groups are unique because they allow you to compress arbitrarily large numbers into group elements, even numbers much larger than the size of the group element, in a way that can't be "spoofed"; constructions from VDFs to [accumulators](#) to range proofs to polynomial commitments can be built on top of this. Another option is to use bulletproofs, using regular elliptic curve groups at the cost of the proof taking much longer to verify. Because polynomial commitments are much simpler than full-on zero knowledge proof schemes, we can expect more such schemes to get created in the future.

Recap

To finish off, let's go over the scheme again. Given a program P, you convert it into a circuit, and generate a set of equations that look like this:

$$ \left(Q_{L_{i}}\right) a_{i}+\left(Q_{R_{i}}\right) b_{i}+\left(Q_{O_{i}}\right) c_{i}+\left(Q_{M_{i}}\right) a_{i} b_{i}+Q_{C_{i}}=0 $$

You then convert this set of equations into a single polynomial equation:

$$ Q_{L}(x) a(x)+Q_{R}(x) b(x)+Q_{O}(x) c(x)+Q_{M}(x) a(x) b(x)+Q_{C}(x)=0 $$

You also generate from the circuit a list of copy constraints. From these copy constraints you generate the three polynomials representing the permuted wire indices: $\sigma_a(x), \sigma_b(x), \sigma_c(x)$. To generate a proof, you compute the values of all the wires and convert them into three polynomials: $a(x), b(x), c(x)$. You also compute six "coordinate pair accumulator" polynomials as part of the permutation-check argument. Finally you compute the cofactors $H_i(x)$.

There is a set of equations between the polynomials that need to be checked; you can do this by making commitments to the polynomials, opening them at some random z (along with proofs that the openings are correct), and running the equations on these evaluations instead of the original polynomials. The proof itself is just a few commitments and openings and can be checked with a few equations. And that's all there is to it!

# The Dawn of Hybrid Layer 2 Protocols

2019 Aug 28

Special thanks to the Plasma Group team for review and feedback

Current approaches to layer 2 scaling - basically, Plasma and state channels - are increasingly moving from theory to practice, but at the same time it is becoming easier to see the inherent challenges in treating these techniques as a fully fledged scaling solution for Ethereum. Ethereum was arguably successful in large part because of its very easy developer experience: you write a program, publish the program, and anyone can interact with it. Designing a state channel or Plasma application, on the other hand, relies on a lot of explicit reasoning about incentives and application-specific development complexity. State channels work well for specific use cases such as repeated payments between the same two parties and two-player games (as successfully implemented in Celer), but more generalized usage is proving challenging. Plasma, particularly Plasma Cash, can work well for payments, but generalization similarly incurs challenges: even implementing a decentralized exchange requires clients to store much more history data, and generalizing to Ethereum-style smart contracts on Plasma seems extremely difficult.

But at the same time, there is a resurgence of a forgotten category of "semi-layer-2" protocols - a category which promises less extreme gains in scaling, but with the benefit of much easier generalization and more favorable security models. A long-forgotten blog post from 2014 introduced the idea of "shadow chains", an architecture where block data is published on-chain, but blocks are not *verified* by default. Rather, blocks are tentatively accepted, and only finalized after some period of time (eg. 2 weeks). During those 2 weeks, a tentatively accepted block can be challenged; only then is the block verified, and if the block proves to be invalid then the chain from that block on is reverted, and the original publisher's deposit is penalized. The contract does not keep track of the full state of the system; it only keeps track of the state root, and users themselves can calculate the state by processing the data submitted to the chain from start to head. A more recent proposal, ZK Rollup, does the same thing without challenge periods, by using ZK-SNARKs to verify blocks' validity.
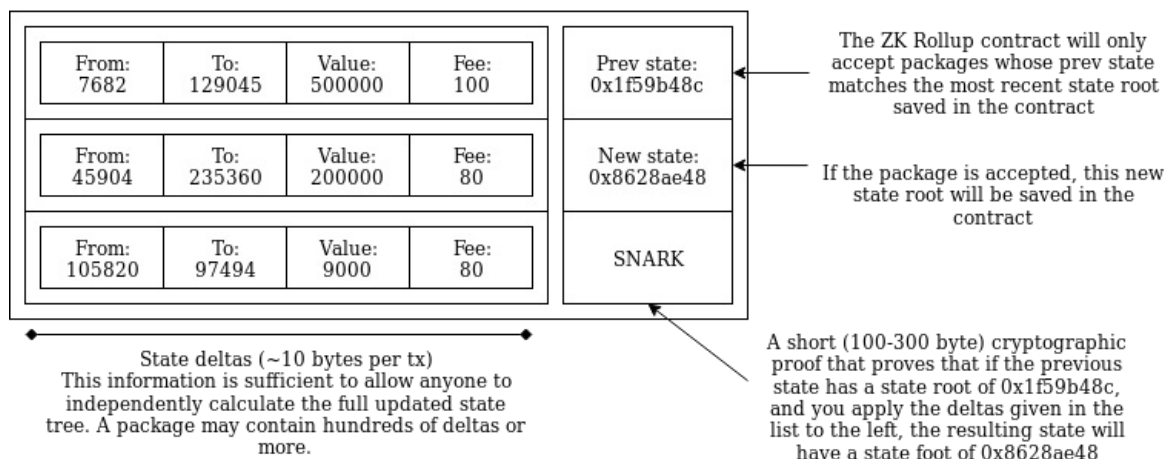
Anatomy of a ZK Rollup package that is published on-chain. Hundreds of "internal transactions" that affect the state (ie. account balances) of the ZK Rollup system are compressed into a package that contains ~10 bytes per internal transaction that specifies the state transitions, plus a ~100-300 byte SNARK proving that the transitions are all valid.

In both cases, the main chain is used to verify data *availability*, but does not (directly) verify block *validity* or perform any significant computation, unless challenges are made. This technique is thus not a jaw-droppingly huge scalability gain, because the on-chain data overhead eventually presents a bottleneck, but it is nevertheless a very significant one. Data is cheaper than computation, and there

are ways to compress transaction data very significantly, particularly because the great majority of data in a transaction is the signature and many signatures can be compressed into one through many forms of aggregation. ZK Rollup promises 500 tx/sec, a 30x gain over the Ethereum chain itself, by compressing each transaction to a mere ~10 bytes; signatures do not need to be included because their validity is verified by the zero-knowledge proof. With BLS aggregate signatures a similar throughput can be achieved in shadow chains (more recently called "optimistic rollup" to highlight its similarities to ZK Rollup). The upcoming [Istanbul hard fork](#) will reduce the gas cost of data from 68 per byte to 16 per byte, increasing the throughput of these techniques by another 4x (that's **over 2000 transactions per second**).

So what is the benefit of data on-chain techniques such as ZK/optimistic rollup versus data off-chain techniques such as Plasma? First of all, there is no need for semi-trusted operators. In ZK Rollup, because validity is verified by cryptographic proofs there is literally no way for a package submitter to be malicious (depending on the setup, a malicious submitter may cause the system to halt for a few seconds, but this is the most harm that can be done). In optimistic rollup, a malicious submitter can publish a bad block, but the next submitter will immediately challenge that block before publishing their own. In both ZK and optimistic rollup, enough data is published on chain to allow anyone to compute the complete internal state, simply by processing all of the submitted deltas in order, and there is no "data withholding attack" that can take this property away. Hence, becoming an operator can be fully permissionless; all that is needed is a security deposit (eg. 10 ETH) for anti-spam purposes.

Second, optimistic rollup particularly is vastly easier to generalize; the state transition function in an optimistic rollup system can be literally anything that can be computed within the gas limit of a single block (including the Merkle branches providing the parts of the state needed to verify the transition). ZK Rollup is theoretically generalizeable in the same way, though in practice making ZK SNARKs over general-purpose computation (such as EVM execution) is very difficult, at least for now. Third, optimistic rollup is much easier to build clients for, as there is less need for second-layer networking infrastructure; more can be done by just scanning the blockchain.

But where do these advantages come from? The answer lies in a highly technical issue known as the *data availability problem* (see [note](#), [video](#)). Basically, there are two ways to try to cheat in a layer-2 system. The first is to publish invalid data to the blockchain. The second is to not publish data at all (eg. in Plasma, publishing the root hash of a new Plasma block to the main chain but without revealing the contents of the block to anyone). Published-but-invalid data is very easy to deal with, because once the data is published on-chain there are multiple ways to figure out unambiguously whether or not it's valid, and an invalid submission is unambiguously invalid so the submitter can be heavily penalized. Unavailable data, on the other hand, is much harder to deal with, because even though unavailability can be detected if challenged, one cannot reliably determine whose fault the non-publication is, especially if data is withheld by default and revealed on-demand only when some verification mechanism tries to verify its availability. This is illustrated in the "Fisherman's dilemma", which shows how a challenge-response game cannot distinguish between malicious submitters and malicious challengers:
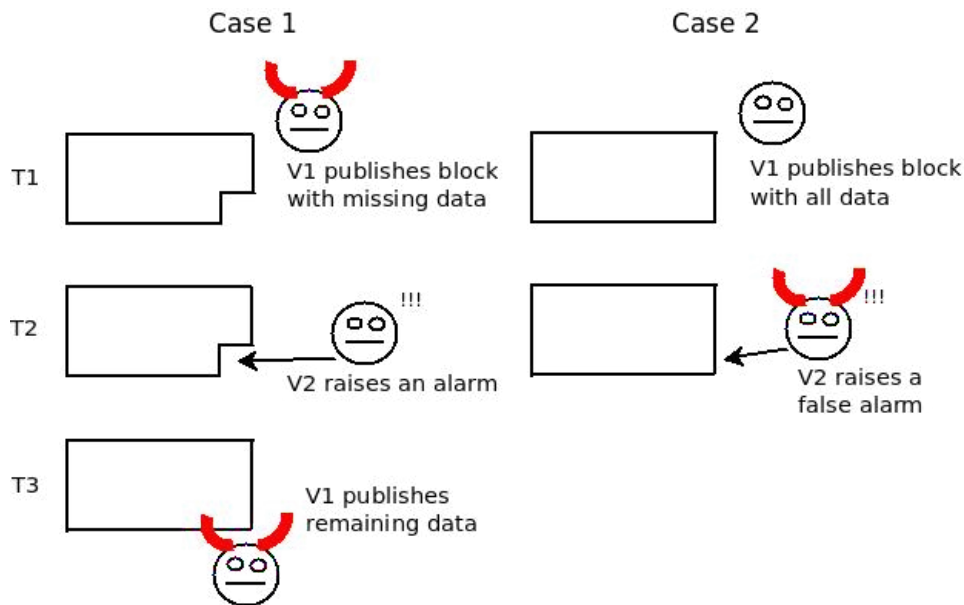
Fisherman's dilemma. If you only start watching the given specific piece of data at time T3, you have no idea whether you are living in Case 1 or Case 2, and hence who is at fault.

Plasma and channels both work around the fisherman's dilemma by pushing the problem to users: if you as a user decide that another user you are interacting with (a counterparty in a state channel, an operator in a Plasma chain) is not publishing data to you that they should be publishing, it's your responsibility to exit and move to a different counterparty/operator. The fact that you as a user have all of the *previous* data, and data about all of the transactions *you* signed, allows you to prove to the chain what assets you held inside the layer-2 protocol, and thus safely bring them out of the system. You prove the existence of a (previously agreed) operation that gave the asset to you, no one else can prove the existence of an operation approved by you that sent the asset to someone else, so you get the asset.

The technique is very elegant. However, it relies on a key assumption: that every state object has a logical "owner", and the state of the object cannot be changed without the owner's consent. This works well for UTXO-based payments (but not account-based payments, where you *can* edit someone else's balance *upward* without their consent; this is why account-based Plasma is so hard), and it can even be made to work for a decentralized exchange, but this "ownership" property is far from universal. Some applications, eg. [Uniswap](#) don't have a natural owner, and even in those applications that do, there are often multiple people that can legitimately make edits to the object. And there is no way to allow arbitrary third parties to exit an asset without introducing the possibility of denial-of-service (DoS) attacks, precisely because one cannot prove whether the publisher or submitter is at fault.

There are other issues peculiar to Plasma and channels individually. Channels do not allow off-chain transactions to users that are not already part of the channel (argument: suppose there existed a way to send $1 to an arbitrary new user from inside a channel. Then this technique could be used many times in parallel to send $1 to more users than there are funds in the system, already breaking its security guarantee). Plasma requires users to store large amounts of history data, which gets even bigger when different assets can be intertwined (eg. when an asset is transferred conditional on transfer of another asset, as happens in a decentralized exchange with a single-stage order book mechanism).

Because data-on-chain computation-off-chain layer 2 techniques don't have data availability issues, they have none of these weaknesses. ZK and optimistic rollup take great care to put enough data on chain to allow users to calculate the full state of the layer 2 system, ensuring that if any participant disappears a new one can trivially take their place. The only issue that they have is verifying computation without doing the computation on-chain, which is a much easier problem. And the scalability gains are significant: ~10 bytes per transaction in ZK Rollup, and a similar level of scalability can be achieved in optimistic rollup by using BLS aggregation to aggregate signatures. This corresponds to a theoretical maximum of ~500 transactions per second today, and over 2000 post-Istanbul.

But what if you want more scalability? Then there is a large middle ground between data-on-chain layer 2 and data-off-chain layer 2 protocols, with many hybrid approaches that give you some of the benefits of both. To give a simple example, the history storage blowup in a decentralized exchange implemented on Plasma Cash can be prevented by publishing a mapping of which orders are matched with which orders (that's less than 4 bytes per order) on chain:
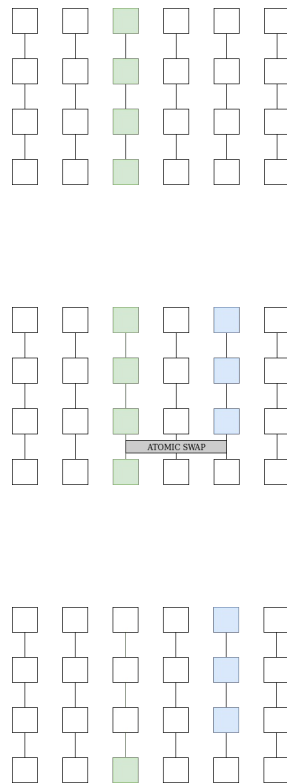
*Left: History data a Plasma Cash user needs to store if they own 1 coin. **Middle:** History data a Plasma Cash user needs to store if they own 1 coin that was exchanged with another coin using an atomic swap. **Right**: History data a Plasma Cash user needs to store if the order matching is published on chain.*

Even outside of the decentralized exchange context, the amount of history that users need to store in Plasma can be reduced by having the Plasma chain periodically publish some per-user data on-chain. One could also imagine a platform which works like Plasma in the case where some state *does* have a logical "owner" and works like ZK or optimistic rollup in the case where it does not. Plasma developers [are already starting to work](#) on these kinds of optimizations.

There is thus a strong case to be made for developers of layer 2 scalability solutions to move to be more willing to publish per-user data on-chain at least some of the time: it greatly increases ease of development, generality and security and reduces per-user load (eg. no need for users storing history data). The efficiency losses of doing so are also overstated: even in a fully off-chain layer-2 architecture, users depositing, withdrawing and moving between different counterparties and providers is going to be an inevitable and frequent occurrence, and so there will be a significant amount of per-user on-chain data regardless. The hybrid route opens the door to a relatively fast deployment of fully generalized Ethereum-style smart contracts inside a quasi-layer-2 architecture.

See also:

- [Introducing the OVM](#)
- [Blog post by Karl Floersch](#)

- [Related ideas by John Adler](#)

Sidechains vs Plasma vs Sharding

Special thanks to Jinglan Wang for review and feedback

One question that often comes up is: how exactly is sharding different from sidechains or Plasma? All three architectures seem to involve a hub-and-spoke architecture with a central "main chain" that serves as the consensus backbone of the system, and a set of "child" chains containing actual user-level transactions. Hashes from the child chains are usually periodically published into the main chain (sharded chains with no hub are theoretically possible but haven't been done so far; this article will not focus on them, but the arguments are similar). Given this fundamental similarity, why go with one approach over the others?

Distinguishing sidechains from Plasma is simple. Plasma chains are sidechains that have a non-custodial property: if there is any error in the Plasma chain, then the error can be detected, and users can safely exit the Plasma chain and prevent the attacker from doing any lasting damage. The only cost that users suffer is that they must wait for a challenge period and pay some higher transaction fees on the (non-scalable) base chain. Regular sidechains do not have this safety property, so they are less secure. However, designing Plasma chains is in many cases much harder, and one could argue that for many low-value applications the security is not worth the added complexity.

So what about Plasma versus sharding? The key technical difference has to do with the notion of **tight coupling**. Tight coupling is a property of sharding, but NOT a property of sidechains or Plasma, that says that the validity of the main chain ("beacon chain" in ethereum 2.0) is inseparable from the validity of the child chains. That is, a child chain block that specifies an invalid main chain block as a dependency is by definition invalid, and more importantly a main chain block that includes an invalid child chain block is by definition invalid.

In non-sharded blockchains, this idea that the canonical chain (ie. the chain that everyone accepts as representing the "real" history) is *by definition* fully available and valid also applies; for example in the case of Bitcoin and Ethereum one typically says that the canonical chain is the "longest valid chain" (or, more pedantically, the "heaviest valid and available chain"). In sharded blockchains, this idea that the canonical chain is the heaviest valid and available chain *by definition* also applies, with the validity and availability requirement applying to both the main chain and shard chains. The new challenge that a sharded system has, however, is that users have no way of fully verifying the validity and availability of any given chain *directly*, because there is too much data. The challenge of engineering sharded chains is to get around this limitation by giving users a maximally trustless and practical *indirect* means to verify which chains are fully available and valid, so that they can still determine which chain is canonical. In practice, this includes techniques like committees, SNARKs/STARKs, fisherman schemes and [fraud and data availability proofs](#).

If a chain structure does not have this tight-coupling property, then it is arguably not a layer-1 sharding scheme, but rather a layer-2 system sitting on top of a non-scalable layer-1 chain. Plasma is not a tightly-coupled system: an invalid Plasma block absolutely can have its header be committed into the main Ethereum chain, because the Ethereum base layer has no idea that it represents an invalid Plasma block, or even that it represents a Plasma block at all; all that it sees is a transaction containing a small piece of data. However, the consequences of a single Plasma chain failing are localized to within that Plasma chain.

> **Sharding** Try really hard to ensure total validity/availability of every part of the system
>
> **Plasma** Accept local faults but try to limit their consequences

However, if you try to analyze the process of *how* users perform the "indirect validation" procedure to determine if the chain they are looking at is fully valid and available without downloading and executing the whole thing, one can find more similarities with how Plasma works. For example, a common technique used to prevent availability issues is fishermen: if a node sees a given piece of a block as unavailable, it can publish a challenge claiming this, creating a time period within which anyone can publish that piece of data. If a block goes unchallenged for long enough, the blocks and

all blocks that cite it as a dependency can be reverted. This seems fundamentally similar to Plasma, where if a block is unavailable users can publish a message to the main chain to exit their state in response. Both techniques eventually buckle under pressure in the same way: if there are too many false challenges in a sharded system, then users cannot keep track of whether or not all of the availability challenges have been answered, and if there are too many availability challenges in a Plasma system then the main chain could get overwhelmed as the exits fill up the chain's block size limit. In both cases, it seems like there's a system that has nominally $O(C^2)$ scalability (where C is the computing power of one node) but where scalability falls to $O(C)$ in the event of an attack. However, sharding has more defenses against this.

First of all, modern sharded designs use randomly sampled committees, so one cannot easily dominate even one committee enough to produce a fake block unless one has a large portion (perhaps $>\frac{1}{3}$) of the entire validator set of the chain. Second, there are better strategies to handling data availability than fishermen: data availability proofs. In a scheme using data availability proofs, if a block is *unavailable*, then clients' data availability checks will fail and clients will see that block as unavailable. If the block is *invalid*, then even a single fraud proof will convince them of this fact for an entire block. An $O(1)$-sized fraud proof can convince a client of the invalidity of an $O(C)$-sized block, and so $O(C)$ data suffices to convince a client of the invalidity of $O(C^2)$ data (this is in the worst case where the client is dealing with N sister blocks all with the same parent of which only one is valid; in more likely cases, one single fraud proof suffices to prove invalidity of an entire invalid chain). Hence, sharded systems are theoretically less vulnerable to being overwhelmed by denial-of-service attacks than Plasma chains.

Second, sharded chains provide stronger guarantees in the face of large and majority attackers (with more than $\frac{1}{3}$ or even $\frac{1}{2}$ of the validator set). A Plasma chain can always be successfully attacked by a 51% attack on the main chain that censors exits; a sharded chain cannot. This is because data availability proofs and fraud proofs happen *inside the client*, rather than *inside the chain*, so they cannot be censored by 51% attacks. Third, the defenses provided by sharded chains are easier to generalize; Plasma's model of exits requires state to be separated into discrete pieces each of which is in the interest of any single actor to maintain, whereas sharded chains relying on data availability proofs, fraud proofs, fishermen and random sampling are theoretically universal.

So there really is a large difference between validity and availability guarantees that are provided at layer 2, which are limited and more complex as they require explicit reasoning about incentives and which party has an interest in which pieces of state, and guarantees that are provided by a layer 1 system that is committed to fully satisfying them.

But Plasma chains also have large advantages too. First, they can be iterated and new designs can be implemented more quickly, as each Plasma chain can be deployed separately without coordinating the rest of the ecosystem. Second, sharding is inherently more fragile, as it attempts to guarantee absolute and total availability and validity of some quantity of data, and this quantity must be set in the protocol; too little, and the system has less scalability than it could have had, too much, and the entire system risks breaking. The maximum safe level of scalability also depends on the number of users of the system, which is an unpredictable variable. Plasma chains, on the other hand, allow different users to make different tradeoffs in this regard, and allow users to adjust more flexibly to changes in circumstances.

Single-operator Plasma chains can also be used to offer more privacy than sharded systems, where all data is public. Even where privacy is not desired, they are potentially more efficient, because the total data availability requirement of sharded systems requires a large extra level of redundancy as a safety margin. In Plasma systems, on the other hand, data requirements for each piece of data can be minimized, to the point where in the long term each individual piece of data may only need to be replicated a few times, rather than a thousand times as is the case in sharded systems.

Hence, in the long term, a hybrid system where a sharded base layer exists, and Plasma chains exist on top of it to provide further scalability, seems like the most likely approach, more able to serve different groups' of users need than sole reliance on one strategy or the other. And it is unfortunately *not* the case that at a sufficient level of advancement Plasma and sharding collapse into the same design; the two are in some key ways irreducibly different (eg. the data availability checks made by clients in sharded systems *cannot* be moved to the main chain in Plasma because these checks only work if they are done subjectively and based on private information). But both scalability solutions (as well as state channels!) have a bright future ahead of them.

Fast Fourier Transforms

2019 May 12

Trigger warning: specialized mathematical topic

Special thanks to Karl Floersch for feedback

One of the more interesting algorithms in number theory is the Fast Fourier transform (FFT). FFTs are a key building block in many algorithms, including extremely fast multiplication of large numbers, multiplication of polynomials, and extremely fast generation and recovery of erasure codes. Erasure codes in particular are highly versatile; in addition to their basic use cases in fault-tolerant data storage and recovery, erasure codes also have more advanced use cases such as securing data availability in scalable blockchains and STARKs. This article will go into what fast Fourier transforms are, and how some of the simpler algorithms for computing them work.

Background

The original Fourier transform is a mathematical operation that is often described as converting data between the "frequency domain" and the "time domain". What this means more precisely is that if you have a piece of data, then running the algorithm would come up with a collection of sine waves with different frequencies and amplitudes that, if you added them together, would approximate the original data. Fourier transforms can be used for such wonderful things as expressing square orbits through epicycles and deriving a set of equations that can draw an elephant:

Ok fine, Fourier transforms also have really important applications in signal processing, quantum mechanics, and other areas, and help make significant parts of the global economy happen. But come on, elephants are cooler.

Running the Fourier transform algorithm in the "inverse" direction would simply take the sine waves and add them together and compute the resulting values at as many points as you wanted to sample.

The kind of Fourier transform we'll be talking about in this post is a similar algorithm, except instead of being a *continuous* Fourier transform over *real or complex numbers*, it's a **discrete Fourier transform** over *finite fields* (see the "A Modular Math Interlude" section here for a refresher on what finite fields are). Instead of talking about converting between "frequency domain" and "time domain", here we'll talk about two different operations: *multi-point polynomial evaluation* (evaluating a degree $< N$ polynomial at N different points) and its inverse, *polynomial interpolation* (given the evaluations of a degree $< N$ polynomial at N different points, recovering the polynomial). For example, if we are operating in the prime field with modulus 5, then the polynomial $y = x^2 + 3$ (for convenience we can write the coefficients in increasing order: $[3,0,1]$) evaluated at the points $[0,1,2]$ gives the values $[3,4,2]$ (not $[3, 4, 7]$ because we're operating in a finite field where the numbers wrap around at 5), and we can actually take the evaluations $[3,4,2]$ and the

coordinates they were evaluated at ($[0,1,2]$) to recover the original polynomial $[3,0,1]$.

There are algorithms for both multi-point evaluation and interpolation that can do either operation in $O(N^2)$ time. Multi-point evaluation is simple: just separately evaluate the polynomial at each point. Here's python code for doing that:

```
def eval_poly_at(self, poly, x, modulus):
    y = 0
    power_of_x = 1
    for coefficient in poly:
        y += power_of_x * coefficient
        power_of_x *= x
    return y % modulus
```

The algorithm runs a loop going through every coefficient and does one thing for each coefficient, so it runs in $O(N)$ time. Multi-point evaluation involves doing this evaluation at N different points, so the total run time is $O(N^2)$.

Lagrange interpolation is more complicated (search for "Lagrange interpolation" here for a more detailed explanation). The key building block of the basic strategy is that for any domain D and point x, we can construct a polynomial that returns 1 for x and 0 for any value in D other than x. For example, if $D = [1,2,3,4]$ and $x = 1$, the polynomial is:

$$ y = \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} $$

You can mentally plug in 1, 2, 3 and 4 to the above expression and verify that it returns 1 for $x= 1$ and 0 in the other three cases.

We can recover the polynomial that gives any desired set of outputs on the given domain by multiplying and adding these polynomials. If we call the above polynomial P_1, and the equivalent ones for $x=2$, $x=3$, $x=4$, P_2, P_3 and P_4, then the polynomial that returns $[3,1,4,1]$ on the domain $[1,2,3,4]$ is simply $3 \cdot P_1 + P_2 + 4 \cdot P_3 + P_4$. Computing the P_i polynomials takes $O(N^2)$ time (you first construct the polynomial that returns to 0 on the entire domain, which takes $O(N^2)$ time, then separately divide it by $(x - x_i)$ for each x_i), and computing the linear combination takes another $O(N^2)$ time, so it's $O(N^2)$ runtime total.

What Fast Fourier transforms let us do, is make both multi-point evaluation and interpolation much faster.

Fast Fourier Transforms

There is a price you have to pay for using this much faster algorithm, which is that you cannot choose any arbitrary field and any arbitrary domain. Whereas with Lagrange interpolation, you could choose whatever x coordinates and y coordinates you wanted, and whatever field you wanted (you could even do it over plain old real numbers), and you could get a polynomial that passes through them., with an FFT, you have to use a finite field, and the domain must be a *multiplicative subgroup* of the field (that is, a list of powers of some "generator" value). For example, you could use the finite field of integers modulo 337, and for the domain use $[1, 85, 148, 111, 336, 252, 189, 226]$ (that's the powers of 85 in the field, eg. 85^3 % $337 = 111$; it stops at 226 because the next power of 85 cycles back to 1). Futhermore, the multiplicative subgroup must have size 2^n (there's ways to make it work for numbers of the form $2^{m} \cdot 3^n$ and possibly slightly higher prime powers but then it gets much more complicated and inefficient). The finite field of intergers modulo 59, for example, would not work, because there are only multiplicative subgroups of order 2, 29 and 58; 2 is too small to be interesting, and the factor 29 is far too large to be FFT-friendly. The symmetry that comes from multiplicative groups of size 2^n lets us create a recursive algorithm that quite cleverly calculate the results we need from a much smaller amount of work.

To understand the algorithm and why it has a low runtime, it's important to understand the general concept of recursion. A recursive algorithm is an algorithm that has two cases: a "base case" where the input to the algorithm is small enough that you can give the output directly, and the "recursive case" where the required computation consists of some "glue computation" plus one or more uses of the same algorithm to smaller inputs. For example, you might have seen recursive algorithms being used for sorting lists. If you have a list (eg. $[1,8,7,4,5,6,3,2,9]$), then you can sort it using the following procedure:

- If the input has one element, then it's already "sorted", so you can just return the input.
- If the input has more than one element, then separately sort the first half of the list and the

second half of the list, and then merge the two sorted sub-lists (call them A and B) as follows. Maintain two counters, $apos$ and $bpos$, both starting at zero, and maintain an output list, which starts empty. Until either $apos$ or $bpos$ is at the end of the corresponding list, check if $A[apos]$ or $B[bpos]$ is smaller. Whichever is smaller, add that value to the end of the output list, and increase that counter by 1. Once this is done, add the rest of whatever list has not been fully processed to the end of the output list, and return the output list.

Note that the "glue" in the second procedure has runtime $O(N)$: if each of the two sub-lists has N elements, then you need to run through every item in each list once, so it's $O(N)$ computation total. So the algorithm as a whole works by taking a problem of size N, and breaking it up into two problems of size $\frac{N}{2}$, plus $O(N)$ of "glue" execution. There is a theorem called the [Master Theorem](link) that lets us compute the total runtime of algorithms like this. It has many sub-cases, but in the case where you break up an execution of size N into k sub-cases of size $\frac{N}{k}$ with $O(N)$ glue (as is the case here), the result is that the execution takes time $O(N \cdot log(N))$.
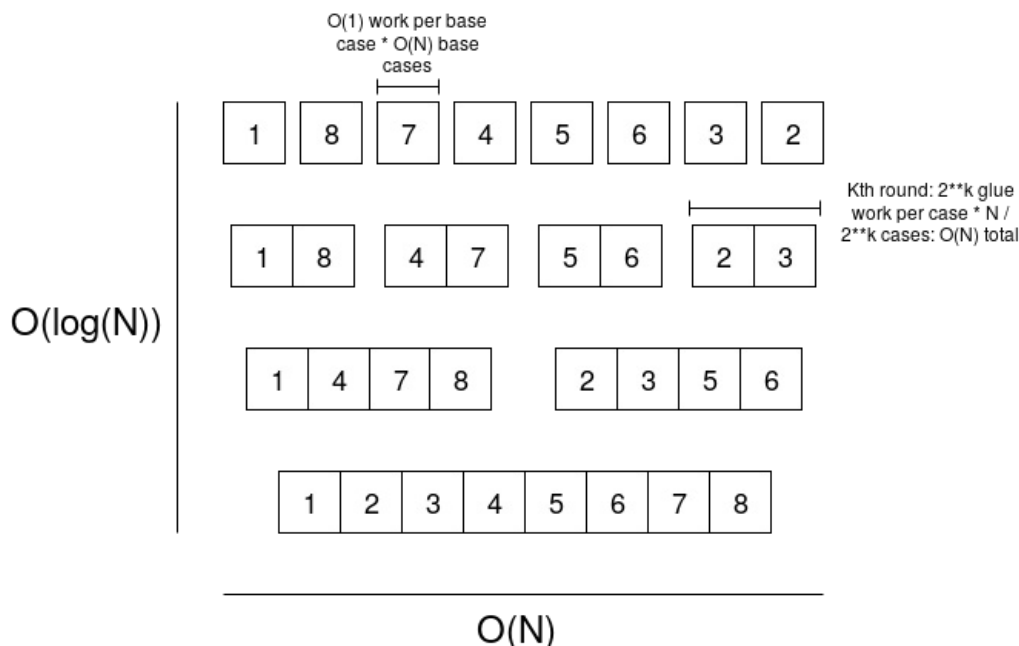
An FFT works in the same way. We take a problem of size N, break it up into two problems of size $\frac{N}{2}$, and do $O(N)$ glue work to combine the smaller solutions into a bigger solution, so we get $O(N \cdot log(N))$ runtime total - *much faster* than $O(N^2)$. Here is how we do it. I'll describe first how to use an FFT for multi-point evaluation (ie. for some domain D and polynomial P, calculate $P(x)$ for every x in D), and it turns out that you can use the same algorithm for interpolation with a minor tweak.

Suppose that we have an FFT where the given domain is the powers of x in some field, where $x^{2^{k}} = 1$ (eg. in the case we introduced above, the domain is the powers of 85 modulo 337, and $85^{2^{3}} = 1$). We have some polynomial, eg. $y = 6x^7 + 2x^6 + 9x^5 + 5x^4 + x^3 + 4x^2 + x + 3$ (we'll write it as $p = [3, 1, 4, 1, 5, 9, 2, 6]$). We want to evaluate this polynomial at each point in the domain, ie. at each of the eight powers of 85. Here is what we do. First, we break up the polynomial into two parts, which we'll call $evens$ and $odds$: $evens = [3, 4, 5, 2]$ and $odds = [1, 1, 9, 6]$ (or $evens = 2x^3 + 5x^2 + 4x + 3$ and $odds = 6x^3 + 9x^2 + x + 1$; yes, this is just taking the even-degree coefficients and the odd-degree coefficients). Now, we note a mathematical observation: $p(x) = evens(x^2) + x \cdot odds(x^2)$ and $p(-x) = evens(x^2) - x \cdot odds(x^2)$ (think about this for yourself and make sure you understand it before going further).

Here, we have a nice property: $evens$ and $odds$ are both polynomials half the size of p, and furthermore, the set of possible values of x^2 is only half the size of the original domain, because there is a two-to-one correspondence: x and $-x$ are both part of D (eg. in our current domain $[1, 85, 148, 111, 336, 252, 189, 226]$, 1 and 336 are negatives of each other, as $336 = -1$ % 337, as are $(85, 252)$, $(148, 189)$ and $(111, 226)$. And x and $-x$ always both have the same square. Hence, we can use an FFT to compute the result of $evens(x)$ for every x in the smaller domain consisting of squares of numbers in the original domain ($[1, 148, 336, 189]$), and we can do the same for odds. And voila, we've reduced a size-N problem into half-size problems.

The "glue" is relatively easy (and $O(N)$ in runtime): we receive the evaluations of $evens$ and $odds$ as size-$\frac{N}{2}$ lists, so we simply do $p[i] = evens\_result[i] + domain[i]\cdot odds\_result[i]$ and $p[\frac{N}{2} + i] = evens\_result[i] - domain[i]\cdot odds\_result[i]$ for each index i.

Here's the full code:

```
def fft(vals, modulus, domain):
    if len(vals) == 1:
        return vals
    L = fft(vals[::2], modulus, domain[::2])
    R = fft(vals[1::2], modulus, domain[::2])
    o = [0 for i in vals]
    for i, (x, y) in enumerate(zip(L, R)):
        y_times_root = y*domain[i]
        o[i] = (x+y_times_root) % modulus
        o[i+len(L)] = (x-y_times_root) % modulus
    return o
```

We can try running it:

```
>>> fft([3,1,4,1,5,9,2,6], 337, [1, 85, 148, 111, 336, 252, 189, 226])
[31, 70, 109, 74, 334, 181, 232, 4]
```

And we can check the result; evaluating the polynomial at the position 85, for example, actually does give the result 70. Note that this only works if the domain is "correct"; it needs to be of the form $[x^i$ % $modulus]$ for i in $range(n)]$ where $x^n = 1$.

An inverse FFT is surprisingly simple:

```
def inverse_fft(vals, modulus, domain):
    vals = fft(vals, modulus, domain)
    return [x * modular_inverse(len(vals), modulus) % modulus for x in [vals[0]] + vals[1:][::-1]]
```

Basically, run the FFT again, but reverse the result (except the first item stays in place) and divide every value by the length of the list.

```
>>> domain = [1, 85, 148, 111, 336, 252, 189, 226]
>>> def modular_inverse(x, n): return pow(x, n - 2, n)
>>> values = fft([3,1,4,1,5,9,2,6], 337, domain)
>>> values
[31, 70, 109, 74, 334, 181, 232, 4]
>>> inverse_fft(values, 337, domain)
[3, 1, 4, 1, 5, 9, 2, 6]
```

Now, what can we use this for? Here's one fun use case: we can use FFTs to multiply numbers very quickly. Suppose we wanted to multiply 1253 by 1895. Here is what we would do. First, we would convert the problem into one that turns out to be slightly easier: multiply the *polynomials* $[3, 5, 2, 1]$ by $[5, 9, 8, 1]$ (that's just the digits of the two numbers in increasing order), and then convert the answer back into a number by doing a single pass to carry over tens digits. We can multiply polynomials with FFTs quickly, because it turns out that if you convert a polynomial into *evaluation form* (ie. $f(x)$ for every x in some domain D), then you can multiply two polynomials simply by multiplying their evaluations. So what we'll do is take the polynomials representing our two numbers in *coefficient form*, use FFTs to convert them to evaluation form, multiply them pointwise, and convert back:

```
>>> p1 = [3,5,2,1,0,0,0,0]
>>> p2 = [5,9,8,1,0,0,0,0]
>>> x1 = fft(p1, 337, domain)
>>> x1
[11, 161, 256, 10, 336, 100, 83, 78]
>>> x2 = fft(p2, 337, domain)
>>> x2
[23, 43, 170, 242, 3, 313, 161, 96]
>>> x3 = [(v1 * v2) % 337 for v1, v2 in zip(x1, x2)]
>>> x3
[253, 183, 47, 61, 334, 296, 220, 74]
>>> inverse_fft(x3, 337, domain)
[15, 52, 79, 66, 30, 10, 1, 0]
```

This requires three FFTs (each $O(N \cdot log(N))$ time) and one pointwise multiplication ($O(N)$)

time), so it takes $O(N \cdot log(N))$ time altogether (technically a little bit more than $O(N \cdot log(N))$, because for very big numbers you would need replace 337 with a bigger modulus and that would make multiplication harder, but close enough). This is *much faster* than schoolbook multiplication, which takes $O(N^2)$ time:

```
    3  5  2  1
   ------------
5 | 15 25 10  5
9 |    27 45 18  9
8 |       24 40 16  8
1 |           3  5  2  1
  ---------------------
    15 52 79 66 30 10  1
```

So now we just take the result, and carry the tens digits over (this is a "walk through the list once and do one thing at each point" algorithm so it takes $O(N)$ time):

```
[15, 52, 79, 66, 30, 10, 1, 0]
[ 5, 53, 79, 66, 30, 10, 1, 0]
[ 5,  3, 84, 66, 30, 10, 1, 0]
[ 5,  3,  4, 74, 30, 10, 1, 0]
[ 5,  3,  4,  4, 37, 10, 1, 0]
[ 5,  3,  4,  4,  7, 13, 1, 0]
[ 5,  3,  4,  4,  7,  3, 2, 0]
```

And if we read the digits from top to bottom, we get 2374435. Let's check the answer....

```
>>> 1253 * 1895
2374435
```

Yay! It worked. In practice, on such small inputs, the difference between $O(N \cdot log(N))$ and $O(N^2)$ isn't *that* large, so schoolbook multiplication is faster than this FFT-based multiplication process just because the algorithm is simpler, but on large inputs it makes a really big difference.

But FFTs are useful not just for multiplying numbers; as mentioned above, polynomial multiplication and multi-point evaluation are crucially important operations in implementing erasure coding, which is a very important technique for building many kinds of redundant fault-tolerant systems. If you like fault tolerance and you like efficiency, FFTs are your friend.

FFTs and binary fields

Prime fields are not the only kind of finite field out there. Another kind of finite field (really a special case of the more general concept of an *extension field*, which are kind of like the finite-field equivalent of complex numbers) are binary fields. In an binary field, each element is expressed as a polynomial where all of the entries are 0 or 1, eg. $x^3 + x + 1$. Adding polynomials is done modulo 2, and subtraction is the same as addition (as $-1 = 1 \bmod 2$). We select some irreducible polynomial as a modulus (eg. $x^4 + x + 1$; $x^4 + 1$ would not work because $x^4 + 1$ can be factored into $(x^2 + 1)\cdot(x^2 + 1)$ so it's not "irreducible"); multiplication is done modulo that modulus. For example, in the binary field mod $x^4 + x + 1$, multiplying $x^2 + 1$ by $x^3 + 1$ would give $x^5 + x^3 + x^2 + 1$ if you just do the multiplication, but $x^5 + x^3 + x^2 + 1 = (x^4 + x + 1)\cdot x + (x^3 + x + 1)$, so the result is the remainder $x^3 + x + 1$.

We can express this example as a multiplication table. First multiply $[1, 0, 0, 1]$ (ie. $x^3 + 1$) by $[1, 0, 1]$ (ie. $x^2 + 1$):

```
    1 0 0 1
   --------
1 | 1 0 0 1
0 |   0 0 0 0
1 |     1 0 0 1
  -----------
    1 0 1 1 0 1
```

The multiplication result contains an x^5 term so we can subtract $(x^4 + x + 1)\cdot x$:

```
    1 0 1 1 0 1
  -   1 1 0 0 1    [(x⁴ + x + 1) shifted right by one to reflect being multipled by x]
  -----------
    1 1 0 1 0 0
```

And we get the result, $[1, 1, 0, 1]$ (or $x^3 + x + 1$).

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	3	1	7	5	11	9	15	13
3	0	3	6	5	12	15	10	9	11	8	13	14	7	4	1	2
4	0	4	8	12	3	7	11	15	6	2	14	10	5	1	13	9
5	0	5	10	15	7	2	13	8	14	11	4	1	9	12	3	6
6	0	6	12	10	11	13	7	1	5	3	9	15	14	8	2	4
7	0	7	14	9	15	8	1	6	13	10	3	4	2	5	12	11
8	0	8	3	11	6	14	5	13	12	4	15	7	10	2	9	1
9	0	9	1	8	2	11	3	10	4	13	5	12	6	15	7	14
10	0	10	7	13	14	4	9	3	15	5	8	2	1	11	6	12
11	0	11	5	14	10	1	15	4	7	12	2	9	13	6	8	3
12	0	12	11	7	5	9	14	2	10	6	1	13	15	3	4	8
13	0	13	9	4	1	12	8	5	2	15	11	6	3	14	10	7
14	0	14	15	1	13	3	2	12	9	7	6	8	4	10	11	5
15	0	15	13	2	9	6	4	11	1	14	12	3	8	7	5	10

Addition and multiplication tables for the binary field mod $(x^4 + x + 1)$. Field elements are expressed as integers converted from binary (eg. $x^3 + x^2 \rightarrow 1100 \rightarrow 12$)

Binary fields are interesting for two reasons. First of all, if you want to erasure-code binary data, then binary fields are really convenient because N bytes of data can be directly encoded as a binary field element, and any binary field elements that you generate by performing computations on it will also be N bytes long. You cannot do this with prime fields because prime fields' size is not exactly a power of two; for example, you could encode every 2 bytes as a number from $(0...65536)$ in the prime field modulo 65537 (which is prime), but if you do an FFT on these values, then the output could contain 65536, which cannot be expressed in two bytes. Second, the fact that addition and subtraction become the same operation, and $1 + 1 = 0$, create some "structure" which leads to some very interesting consequences. One particularly interesting, and useful, oddity of binary fields is the "[freshman's dream](#)" theorem: $(x+y)^2 = x^2 + y^2$ (and the same for exponents $4, 8, 16...$ basically any power of two).

But if you want to use binary fields for erasure coding, and do so efficiently, then you need to be able to do Fast Fourier transforms over binary fields. But then there is a problem: in a binary field, *there are no (nontrivial) multiplicative groups of order (2^n)*. This is because the multiplicative groups are all order 2^n-1. For example, in the binary field with modulus $(x^4 + x + 1)$, if you start calculating successive powers of $(x+1)$, you cycle back to 1 after $\it 15$ steps - not 16. The reason is that the total number of elements in the field is 16, but one of them is zero, and you're never going to reach zero by multiplying any nonzero value by itself in a field, so the powers of $(x+1)$ cycle through every element but zero, so the cycle length is 15, not 16. So what do we do?

The reason we needed the domain to have the "structure" of a multiplicative group with (2^n) elements before is that we needed to reduce the size of the domain by a factor of two by squaring each number in it: the domain $([1, 85, 148, 111, 336, 252, 189, 226])$ gets reduced to $([1, 148, 336, 189])$ because 1 is the square of both 1 and 336, 148 is the square of both 85 and (252), and so forth. But what if in a binary field there's a different way to halve the size of a domain? It turns out that there is: given a domain containing (2^k) values, including zero (technically the domain must be a [subspace](#)), we can construct a half-sized new domain D' by taking $(x \cdot (x+k))$ for x in D using some specific k in D. Because the original domain is a subspace, since k is in the domain, any x in the domain has a corresponding $(x+k)$ also in the domain, and the function $f(x) = x \cdot (x+k)$ returns the same value for x and $(x+k)$ so we get the same kind of two-to-one correspondence that squaring gives us.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x \cdot (x+1)$	0	0	6	6	7	7	1	1	4	4	2	2	3	3	5	5

So now, how do we do an FFT on top of this? We'll use the same trick, converting a problem with an N-sized polynomial and N-sized domain into two problems each with an $\frac{N}{2}$-sized polynomial and $\frac{N}{2}$-sized domain, but this time using different equations. We'll convert a polynomial p into two polynomials $evens$ and $odds$ such that $p(x) = evens(x \cdot (k-x)) + x$

$\cdot$ odds(x $\cdot$ (k-x)))$. Note that for the $evens$ and $odds$ that we find, it will *also* be true that $p(x+k) = evens(x \cdot (k-x)) + (x+k) \cdot odds(x \cdot (k-x))$. So we can then recursively do an FFT to $evens$ and $odds$ on the reduced domain $[x \cdot (k-x)]$ for x in D, and then we use these two formulas to get the answers for two "halves" of the domain, one offset by k from the other.

Converting p into $evens$ and $odds$ as described above turns out to itself be nontrivial. The "naive" algorithm for doing this is itself $O(N^2)$, but it turns out that in a binary field, we can use the fact that $(x^2-kx)^2 = x^4 - k^2 \cdot x^2$, and more generally $(x^2-kx)^{2^i} = x^{2^{i+1}} - k^{2^i} \cdot x^{2^i}$, to create yet another recursive algorithm to do this in $O(N \cdot log(N))$ time.

And if you want to do an *inverse* FFT, to do interpolation, then you need to run the steps in the algorithm in reverse order. You can find the complete code for doing this here: https://github.com/ethereum/research/tree/master/binary_fft, and a paper with details on more optimal algorithms here: http://www.math.clemson.edu/~sgao/papers/GM10.pdf

So what do we get from all of this complexity? Well, we can try running the implementation, which features both a "naive" $O(N^2)$ multi-point evaluation and the optimized FFT-based one, and time both. Here are my results:

```
>>> import binary_fft as b
>>> import time, random
>>> f = b.BinaryField(1033)
>>> poly = [random.randrange(1024) for i in range(1024)]
>>> a = time.time(); x1 = b._simple_ft(f, poly); time.time() - a
0.5752472877502441
>>> a = time.time(); x2 = b.fft(f, poly, list(range(1024))); time.time() - a
0.03820443153381348
```

And as the size of the polynomial gets larger, the naive implementation (`_simple_ft`) gets slower much more quickly than the FFT:

```
>>> f = b.BinaryField(2053)
>>> poly = [random.randrange(2048) for i in range(2048)]
>>> a = time.time(); x1 = b._simple_ft(f, poly); time.time() - a
2.2243144512176514
>>> a = time.time(); x2 = b.fft(f, poly, list(range(2048))); time.time() - a
0.07896280288696289
```

And voila, we have an efficient, scalable way to multi-point evaluate and interpolate polynomials. If we want to use FFTs to recover erasure-coded data where we are *missing* some pieces, then algorithms for this also exist, though they are somewhat less efficient than just doing a single FFT. Enjoy!

Control as Liability

2019 May 09

The regulatory and legal environment around internet-based services and applications has changed considerably over the last decade. When large-scale social networking platforms first became popular in the 2000s, the general attitude toward mass data collection was essentially "why not?". This was the age of Mark Zuckerberg saying the age of privacy is over and Eric Schmidt arguing, "If you have something that you don't want anyone to know, maybe you shouldn't be doing it in the first place." And it made personal sense for them to argue this: every bit of data you can get about others was a potential machine learning advantage for you, every single restriction a weakness, and if something happened to that data, the costs were relatively minor. Ten years later, things are very different.

It is especially worth zooming in on a few particular trends.

- **Privacy**. Over the last ten years, a number of privacy laws have been passed, most aggressively in Europe but also elsewhere, but the most recent is the GDPR. The GDPR has many parts, but among the most prominent are: (i) requirements for explicit consent, (ii) requirement to have a legal basis to process data, (iii) users' right to download all their data, (iv) users' right to require you to delete all their data. Other jurisdictions are exploring similar rules.
- **Data localization rules**. India, Russia and many other jurisdictions increasingly have or are exploring rules that require data on users within the country to be stored inside the country. And even when explicit laws do not exist, there's a growing shift toward concern (eg. 1 2) around data being moved to countries that are perceived to not sufficiently protect it.
- **Sharing economy regulation**. Sharing economy companies such as Uber are having a hard time arguing to courts that, given the extent to which their applications control and direct drivers' activity, they should not be legally classified as employers.
- **Cryptocurrency regulation**. A recent FINCEN guidance attempts to clarify what categories of cryptocurrency-related activity are and are not subject to regulatory licensing requirements in the United States. Running a hosted wallet? Regulated. Running a wallet where the user controls their funds? Not regulated. Running an anonymizing mixing service? If you're *running* it, regulated. If you're just writing code... *not regulated*.

As Emin Gun Sirer points out, the FINCEN cryptocurrency guidance is not at all haphazard; rather, it's trying to separate out categories of applications where the developer is actively controlling funds, from applications where the developer has no control. The guidance carefully separates out how *multisignature wallets*, where keys are held both by the operator and the user, are sometimes regulated and sometimes not:

> If the multiple-signature wallet provider restricts its role to creating un-hosted wallets that require adding a second authorization key to the wallet owner's private key in order to validate and complete transactions, the provider is not a money transmitter because it does not accept and transmit value. On the other hand, if ... the value is represented as an entry in the accounts of the provider, the owner does not interact with the payment system directly, or the provider maintains total independent control of the value, the provider will also qualify as a money transmitter.

Although these events are taking place across a variety of contexts and industries, I would argue that there is a common trend at play. And the trend is this: **control over users' data and digital possessions and activity is rapidly moving from an asset to a liability**. Before, every bit of control you have was good: it gives you more flexibility to earn revenue, if not now then in the future. Now, every bit of control you have is a liability: you might be regulated because of it. If you exhibit control over your users' cryptocurrency, you are a money transmitter. If you have "sole discretion over fares, and can charge drivers a cancellation fee if they choose not to take a ride, prohibit drivers from picking up passengers not using the app and suspend or deactivate drivers' accounts", you are an employer. If you control your users' data, you're required to make sure you can argue just cause, have a compliance officer, and give your users access to download or delete the data.

If you are an application builder, and you are both lazy and fear legal trouble, there is one easy way to make sure that you violate none of the above new rules: *don't build applications that centralize control*. If you build a wallet where the user holds their private keys, you really are still "just a

software provider". If you build a "decentralized Uber" that really is just a slick UI combining a payment system, a reputation system and a search engine, and don't control the components yourself, you really won't get hit by many of the same legal issues. If you build a website that just... doesn't collect data (Static web pages? But that's impossible!) you don't have to even think about the GDPR.

This kind of approach is of course not realistic for everyone. There will continue to be many cases where going without the conveniences of centralized control simply sacrifices too much for both developers and users, and there are also cases where the business model considerations mandate a more centralized approach (eg. it's easier to prevent non-paying users from using software if the software stays on your servers) win out. But we're definitely very far from having explored the full range of possibilities that more decentralized approaches offer.

Generally, unintended consequences of laws, discouraging entire categories of activity when one wanted to only surgically forbid a few specific things, are considered to be a bad thing. Here though, I would argue that the forced shift in developers' mindsets, from "I want to control more things just in case" to "I want to control fewer things just in case", also has many positive consequences. Voluntarily giving up control, and voluntarily taking steps to deprive oneself of the ability to do mischief, does not come naturally to many people, and while ideologically-driven decentralization-maximizing projects exist today, it's not at all obvious at first glance that such services will continue to dominate as the industry mainstreams. What this trend in regulation does, however, is that it gives a big nudge in favor of those applications that are willing to take the centralization-minimizing, user-sovereignty-maximizing "can't be evil" route.

Hence, even though these regulatory changes are arguably not pro-freedom, at least if one is concerned with the freedom of application developers, and the transformation of the internet into a subject of political focus is bound to have many negative knock-on effects, the particular trend of control becoming a liability is in a strange way *even more pro-cypherpunk* (even if not intentionally!) than policies of maximizing total freedom for application developers would have been. Though the present-day regulatory landscape is very far from an optimal one from the point of view of almost anyone's preferences, it has unintentionally dealt the movement for minimizing unneeded centralization and maximizing users' control of their own assets, private keys and data a surprisingly strong hand to execute on its vision. And it would be highly beneficial to the movement to take advantage of it.

On Free Speech

2019 Apr 16

"*A statement may be both true and dangerous. The previous sentence is such a statement.*" - *David Friedman*

Freedom of speech is a topic that many internet communities have struggled with over the last two decades. Cryptocurrency and blockchain communities, a major part of their raison d'etre being censorship resistance, are especially poised to value free speech very highly, and yet, over the last few years, the extremely rapid growth of these communities and the very high financial and social stakes involved have repeatedly tested the application and the limits of the concept. In this post, I aim to disentangle some of the contradictions, and make a case what the norm of "free speech" really stands for.

"Free speech laws" vs "free speech"

A common, and in my own view frustrating, argument that I often hear is that "freedom of speech" is exclusively a legal restriction on what *governments* can act against, and has nothing to say regarding the actions of private entities such as corporations, privately-owned platforms, internet forums and conferences. One of the larger examples of "private censorship" in cryptocurrency communities was the decision of Theymos, the moderator of the [/r/bitcoin](#) subreddit, to start heavily moderating the subreddit, forbidding arguments in favor of increasing the Bitcoin blockchain's transaction capacity via a hard fork.

> [–] theymos -45 points 1 year ago*
>
> You can promote BIP 101 as an idea. You can't promote (on /r/Bitcoin) the actual *usage* of BIP 101. When the idea has consensus, *then* it can be rolled out.
>
> Bitcoin is not a democracy. Not of miners, and not of nodes. Switching to XT is not a vote for BIP 101 -- it is abandoning Bitcoin for a separate network/currency. It is good that you have the freedom to do this. One of the great things about Bitcoin *is* its lack of democracy: even if 99% of people use Bitcoin, you are free to implement BIP 101 in a separate currency without the Bitcoin users being able to democratically coerce you into using the real Bitcoin network/currency again. But I am not obligated to allow these separate offshoots of Bitcoin to exist on /r/Bitcoin, and I'm not going to.

Here is a timeline of the censorship as catalogued by John Blocke: [https://medium.com/johnblocke/a-brief-and-incomplete-history-of-censorship-in-r-bitcoin-c85a290fe43](#)

Here is Theymos's post defending his policies: [https://www.reddit.com/r/Bitcoin/comments/3h9cq4/its_time_for_a_break_about_the_recent_mess/](#), including the now infamous line "If 90% of /r/Bitcoin users find these policies to be intolerable, then I want these 90% of /r/Bitcoin users to leave".

A common strategy used by defenders of Theymos's censorship was to say that heavy-handed moderation is okay because /r/bitcoin is "a private forum" owned by Theymos, and so he has the right to do whatever he wants in it; those who dislike it should move to other forums:

> **Party Timez** @PartehTiemz · 3 Feb 2017
> Replying to @adam3us
> I hope Core team can undo the damage caused to it by Theymos. Bitcoin can rely on decentralization, what can the community rely on?
> ♡ 1   ↻   ♡ 1   ✉

> **Neo M. Hodlonaut** 🚀⚡ @RedPillTrading · 4 Feb 2017
> Theymos moderates a PRIVATE forum, nothing to do with core. The community relies on bitcoin. So, let's keep it decentralized.
> ♡ 1   ↻   ♡   ✉

> ▲ beaner 6 months ago [-]
>
> Bitcoin cash isn't censored. It has its own subreddit (and the rest of the internet) where discussion can be h
>
> Equating "censored in r/bitcoin" with censorship in general sort of proves that it's mostly about politics; you uncensored _in a specific private community_. If BCH can stand on its own merit (and hopefully it can!) ther need that. Those who think it does need that aren't trying to make BCH successful, they want to control Bit it makes sense that people with those motives should not be allowed.
>
> Layer 2 is a scaling solution, I don't see why it wouldn't be.

And it's true that Theymos has not *broken any laws* by moderating his forum in this way. But to most people, it's clear that there is still some kind of free speech violation going on. So what gives? First of all, it's crucially important to recognize that freedom of speech is not just a *law in some countries*. It's also a social principle. And the underlying goal of the social principle is the same as the underlying goal of the law: to foster an environment where the ideas that win are ideas that are good, rather than just ideas that happen to be favored by people in a position of power. And governmental power is not the only kind of power that we need to protect from; there is also a corporation's power to fire someone, an internet forum moderator's power to [delete almost every post in a discussion thread](#), and many other kinds of power hard and soft.

So what is the underlying social principle here? [Quoting Eliezer Yudkowsky](#):

> There are a very few injunctions in the human art of rationality that have no ifs, ands, buts, or escape clauses. This is one of them. Bad argument gets counterargument. Does not get bullet. Never. Never ever never for ever.

[Slatestarcodex elaborates](#):

> What does "bullet" mean in the quote above? Are other projectiles covered? Arrows? Boulders launched from catapults? What about melee weapons like swords or maces? Where exactly do we draw the line for "inappropriate responses to an argument"? A good response to an argument is one that addresses an idea; a bad argument is one that silences it. If you try to address an idea, your success depends on how good the idea is; if you try to silence it, your success depends on how powerful you are and how many pitchforks and torches you can provide on short notice. Shooting bullets is a good way to silence an idea without addressing it. So is firing stones from catapults, or slicing people open with swords, or gathering a pitchfork-wielding mob. But trying to get someone fired for holding an idea is also a way of silencing an idea without addressing it.

That said, sometimes there is a rationale for "safe spaces" where people who, for whatever reason, just don't want to deal with arguments of a particular type, can congregate and where those arguments actually do get silenced. Perhaps the most innocuous of all is spaces like [ethresear.ch](#) where posts get silenced just for being "off topic" to keep the discussion focused. But there's also a dark side to the concept of "safe spaces"; as [Ken White writes](#):

> This may come as a surprise, but I'm a supporter of 'safe spaces.' I support safe spaces because I support freedom of association. Safe spaces, if designed in a principled way, are just an application of that freedom... But not everyone imagines "safe spaces" like that. Some use the concept of "safe spaces" as a sword, wielded to annex public spaces and demand that people within those spaces conform to their private norms. That's not freedom of association

Aha. So making your own safe space off in a corner is totally fine, but there is also this concept of a "public space", and trying to turn a public space into a safe space for one particular special interest is wrong. So what is a "public space"? It's definitely clear that a public space is *not* just "a space owned and/or run by a

government"; the concept of [privately owned public spaces](#) is a well-established one. This is true even informally: it's a common moral intuition, for example, that it's less bad for a private individual to commit violations such as discriminating against races and genders than it is for, say, a shopping mall to do the same. In the case or the /r/bitcoin subreddit, one can make the case, regardless of who technically owns the top moderator position in the subreddit, that the subreddit very much is a public space. A few arguments particularly stand out:

- It occupies "prime real estate", specifically the word "bitcoin", which makes people consider it to be *the* default place to discuss Bitcoin.
- The value of the space was created not just by Theymos, but by thousands of people who arrived on the subreddit to discuss Bitcoin with an implicit expectation that it is, and will continue, to be a public space for discussing Bitcoin.
- Theymos's shift in policy was a surprise to many people, and it was *not* foreseeable ahead of time that it would take place.

If, instead, Theymos had created a subreddit called /r/bitcoinsmallblockers, and explicitly said that it was a curated space for small block proponents and attempting to instigate controversial hard forks was not welcome, then it seems likely that very few people would have seen anything wrong about this. They would have opposed his ideology, but few (at least in blockchain communities) would try to claim that it's *improper* for people with ideologies opposed to their own to have spaces for internal discussion. But back in reality, Theymos tried to "annex a public space and demand that people within the space confirm to his private norms", and so we have the Bitcoin community block size schism, a highly acrimonious fork and chain split, and now a cold peace between Bitcoin and Bitcoin Cash.

Deplatforming

About a year ago at Deconomy I publicly shouted down Craig Wright, [a scammer claiming to be Satoshi Nakamoto](#), finishing my explanation of why the things he says make no sense with the question "why is this fraud allowed to speak at this conference?"

Of course, Craig Wright's partisans replied back with.... [accusations of censorship](#):

> "Why do we allow people like Craig Wright to speak at a conference like this?" Buterin then suggested that Dr. Wight's university degrees are not real.
>
> The question was shocking enough, but more shocking was Mow publicly agreeing to this call for censorship. Censorship of others' opinions is exactly what Blockstream and Core stand accused of by many – including directly by Roger in his debate – and here was Mow, in front of the entire crowd, advocating for the silencing of someone's voice.

Did I try to "silence" Craig Wright? I would argue, no. One could argue that this is because "Deconomy is not a public space", but I think the much better argument is that a conference is fundamentally different from an internet forum. An internet forum can actually try to be a fully neutral medium for discussion where anything goes; a conference, on the other hand, is by its very nature a highly curated list of presentations, allocating a limited number of speaking slots and actively channeling a large amount of attention to those lucky enough to get a chance to speak. A conference is an editorial act by the organizers, saying "here are some ideas and views that we think people really should be exposed to and hear". Every conference "censors" almost every viewpoint because there's not enough space to give them all a chance to speak, and this is inherent to the format; so raising an objection to a conference's judgement in making its selections is absolutely a legitimate act.

This extends to other kinds of selective platforms. Online platforms such as Facebook, Twitter and Youtube already engage in active selection through algorithms that influence what people are more likely to be recommended. Typically, they do this for selfish reasons, setting up their algorithms to maximize "engagement" with their platform, often with unintended byproducts like [promoting flat earth conspiracy theories](#). So given that these platforms are already engaging in (automated) selective presentation, it seems eminently reasonable to criticize them for not directing these same levers toward more pro-social objectives, or at the least pro-social objectives that all major reasonable political tribes agree on (eg. quality intellectual discourse). Additionally, the "censorship" doesn't seriously block anyone's ability to learn Craig Wright's side of the story; you can just go visit their website, here you go: https://coingeek.com/. **If someone is already operating a platform that makes editorial decisions, asking them to make such decisions with the same magnitude but with more pro-social criteria seems like a very reasonable thing to do**.

A more recent example of this principle at work is the #DelistBSV campaign, where some cryptocurrency exchanges, most famously [Binance](#), removed support for trading BSV (the Bitcoin fork promoted by Craig Weight). Once again, many people, even [reasonable people](#), accused this campaign of being an [exercise in censorship](#), raising parallels to credit card companies blocking Wikileaks:

What this phenomenon suggests is that the #crypto community's commitment to 'censorship-resistance' and getting rid of human agency/discretion may be about having the power to make the decisions to censor or not.

Power transfer, rather than power distribution.

3:43 PM - 15 Apr 2019

8 Retweets 39 Likes

○ 12   ⇄ 8   ♡ 39   ✉
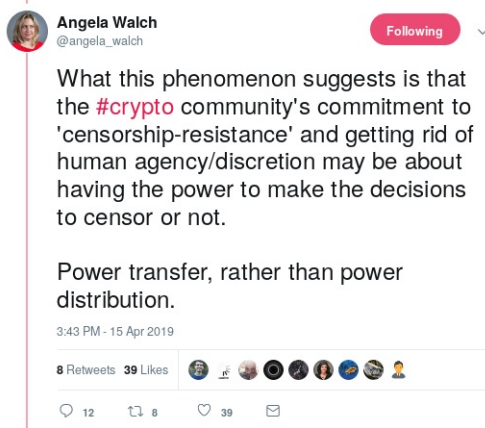
I personally have been a critic of the power wielded by centralized exchanges. Should I oppose #DelistBSV on free speech grounds? I would argue no, it's ok to support it, but this is definitely a much closer call.

Many #DelistBSV participants like Kraken are definitely not "anything-goes" platforms; they already make many editorial decisions about which currencies they accept and refuse. Kraken only accepts about a dozen currencies, so they are passively "censoring" almost everyone. Shapeshift supports more currencies but it does not support SPANK, or even KNC. So in these two cases, delisting BSV is more like reallocation of a scarce resource (attention/legitimacy) than it is censorship. Binance is a bit different; it does accept a very large array of cryptocurrencies, adopting a philosophy much closer to anything-goes, and it does have a unique position as market leader with a lot of liquidity.

That said, one can argue two things in Binance's favor. First of all, censorship is retaliating against a truly malicious exercise of censorship on the part of core BSV community members when they threatened critics like Peter McCormack with legal letters (see Peter's response); in "anarchic" environments with large disagreements on what the norms are, "an eye for an eye" in-kind retaliation is one of the better social norms to have because it ensures that people only face punishments that they in some sense have through their own actions demonstrated they believe are legitimate. Furthermore, the delistings won't make it that hard for people to buy or sell BSV; Coinex has said that they will not delist (and I would actually oppose second-tier "anything-goes" exchanges delisting). But the delistings *do* send a strong message of social condemnation of BSV, which is useful and needed. So there's a case to support all delistings so far, though on reflection Binance refusing to delist "because freedom" would have also been not as unreasonable as it seems at first glance.

It's in general absolutely potentially reasonable to oppose the existence of a concentration of power, but support that concentration of power being used for purposes that you consider prosocial as long as that concentration exists; see Bryan Caplan's exposition on reconciling supporting open borders and also supporting anti-ebola restrictions for an example in a different field. Opposing concentrations of power only requires that one believe those concentrations of power to be *on balance* harmful and abusive; it does not mean that one must oppose *all* things that those concentrations of power do.

If someone manages to make a *completely permissionless* cross-chain decentralized exchange that facilitates trade between any asset and any other asset, then being "listed" on the exchange would *not* send a social signal, because everyone is listed; and I would support such an exchange existing even if it supports trading BSV. The thing that I do support is BSV being removed from already exclusive positions that confer higher tiers of legitimacy than simple existence.

So to conclude: censorship in public spaces bad, even if the public spaces are non-governmental; censorship in genuinely private spaces (especially spaces that are *not* "defaults" for a broader community) can be okay; ostracizing projects with the goal and effect of denying access to them, bad; ostracizing projects with the goal and effect of denying them scarce legitimacy can be okay.

On Collusion

2019 Apr 03

Special thanks to Glen Weyl, Phil Daian and Jinglan Wang for review

Over the last few years there has been an increasing interest in using deliberately engineered economic incentives and mechanism design to align behavior of participants in various contexts. In the blockchain space, mechanism design first and foremost provides the security for the blockchain itself, encouraging miners or proof of stake validators to participate honestly, but more recently it is being applied in prediction markets, "token curated registries" and many other contexts. The nascent RadicalXChange movement has meanwhile spawned experimentation with Harberger taxes, quadratic voting, quadratic financing and more. More recently, there has also been growing interest in using token-based incentives to try to encourage quality posts in social media. However, as development of these systems moves closer from theory to practice, there are a number of challenges that need to be addressed, challenges that I would argue have not yet been adequately confronted.

As a recent example of this move from theory toward deployment, Bihu, a Chinese platform that has recently released a coin-based mechanism for encouraging people to write posts. The basic mechanism (see whitepaper in Chinese here) is that if a user of the platform holds KEY tokens, they have the ability to stake those KEY tokens on articles; every user can make k "upvotes" per day, and the "weight" of each upvote is proportional to the stake of the user making the upvote. Articles with a greater quantity of stake upvoting them appear more prominently, and the author of an article gets a reward of KEY tokens roughly proportional to the quantity of KEY upvoting that article. This is an oversimplification and the actual mechanism has some nonlinearities baked into it, but they are not essential to the basic functioning of the mechanism. KEY has value because it can be used in various ways inside the platform, but particularly a percentage of all ad revenues get used to buy and burn KEY (yay, big thumbs up to them for doing this and not making yet another medium of exchange token!).

This kind of design is far from unique; incentivizing online content creation is something that very many people care about, and there have been many designs of a similar character, as well as some fairly different designs. And in this case this particular platform is already being used significantly:

A few months ago, the Ethereum trading subreddit /r/ethtrader introduced a somewhat similar experimental feature where a token called "donuts" is issued to users that make comments that get upvoted, with a set amount of donuts issued weekly to users in proportion to how many upvotes their comments received. The donuts could be used to buy the right to set the contents of the banner at the top of the subreddit, and could also be used to vote in community polls. However, unlike what happens in the KEY system, here the reward that B receives when A upvotes B is not proportional to A's existing coin supply; instead, each Reddit account

has an equal ability to contribute to other Reddit accounts.
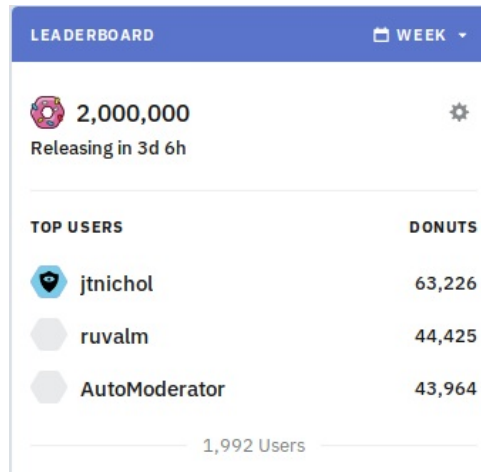
These kinds of experiments, attempting to reward quality content creation in a way that goes beyond the known limitations of donations/microtipping, are very valuable; under-compensation of user-generated internet content is a very significant problem in society in general (see "liberal radicalism" and "data as labor"), and it's heartening to see crypto communities attempting to use the power of mechanism design to make inroads on solving it. **But unfortunately, these systems are also vulnerable to attack.**

Self-voting, plutocracy and bribes

Here is how one might economically attack the design proposed above. Suppose that some wealthy user acquires some quantity N of tokens, and as a result each of the user's k upvotes gives the recipient a reward of $N \cdot q$ (q here probably being a very small number, eg. think $q = 0.000001$). The user simply upvotes their own sockpuppet accounts, giving themselves the reward of $N \cdot k \cdot q$. Then, the system simply collapses into each user having an "interest rate" of $k \cdot q$ per period, and the mechanism accomplishes nothing else.

The actual Bihu mechanism seemed to anticipate this, and has some superlinear logic where articles with more KEY upvoting them gain a disproportionately greater reward, seemingly to encourage upvoting popular posts rather than self-upvoting. It's a common pattern among coin voting governance systems to add this kind of superlinearity to prevent self-voting from undermining the entire system; most DPOS schemes have a limited number of delegate slots with zero rewards for anyone who does not get enough votes to join one of the slots, with similar effect. But these schemes invariably introduce two new weaknesses:

- They **subsidize plutocracy**, as very wealthy individuals and cartels can still get enough funds to self-upvote.
- They can be circumvented by users *bribing* other users to vote for them en masse.

Bribing attacks may sound farfetched (who here has ever accepted a bribe in real life?), but in a mature ecosystem they are much more realistic than they seem. In most contexts where bribing has taken place in the blockchain space, the operators use a euphemistic new name to give the concept a friendly face: it's not a bribe, it's a "staking pool" that "shares dividends". Bribes can even be obfuscated: imagine a cryptocurrency exchange that offers zero fees and spends the effort to make an abnormally good user interface, and does not even try to collect a profit; instead, it uses coins that users deposit to participate in various coin voting systems. There will also inevitably be people that see in-group collusion as just plain normal; see a recent scandal involving EOS DPOS for one example:

火币投票节点	火币投票数	对方回投数	火币投票数	对方回投数	火币投票数	对方回投数
节点名称	9月4日		9月5日		9月10日	
eoshuobipool	1400		1400		1400	
starteosiobp	1000	1300	1000	1300	1200	1400
zbeosbp11111	1400	1500	1400	1500	1400	3705
eosflytomars	700	678	700	678	1700	2142
eostitanprod	200	456	200	440	200	484
bitfinexeos1	1000	4750	1000	4750	1000	4800
eosgenblockp	1400		1400		2000	
eoscannonchn			800	1490	1100	2007
eosfishrocks	300	318	300	318	300	458
eosstorebest	400	200	700	200	700	200
eosbeijingbp	600		600		600	
eosbixinboot	500	200	900	200	900	300
jedaaaaaaaa	500	300	500	300	500	759
eoshenzhenio	500	50	500	50	500	98
eoseouldotio	500		500		500	
atticlabeosb		500		500		500
sheleaders21	500		500		500	527
eospacificbp					2000	
eoslaomaoccm					200	
qxeosqxeosbp					150	280
eoscybexiobp					150	272
geosoneforbp					100	112
cryptokylin1	500		500		500	
eosiosg11111	1400		1400		2000	
cochainworld	1400		1400		2000	
eospaceioeos	1400		1400		2000	
总计	15600	10252	17100	11726	23600	18044

○ 2   ⟲ 14   ♡ 50   ✉

EOS节点每日收入情况	9月5日	9月6日
eoshuobipool	848.7496	830.7248
cryptokylin1	/	/
eosiosg11111	137.3051	257.6671
cochainworld	129.5332	141.9138
eospaceioeos		
截止9月4日累计收入个数		
火币每日总计收入个数		235
折合USDT	4740.74225	494Y...

（注：施霏霏:扣除170个，其他作为我们的收入。）

EOS节点每日收入情况	9月5日	9月6日
eoshuobipool	848.7496	830.7248
cryptokylin1	/	/
eosiosg11111	137.3051	257.6671
cochainworld		.9138
eospaceioeos		9.904
截止9月4日累计收入个数		
火币每日总计收入个数	948.14845	989.46235
折合USDT	4740.74225	494Y...

（注：施霏霏:扣除150个，其余作为我们的收入）

○ 3   ⟲ 10   ♡ 36   ✉

Finally, there is the possibility of a "negative bribe", ie. blackmail or coercion, threatening participants with harm unless they act inside the mechanism in a certain way.

In the /r/ethtrader experiment, fear of people coming in and *buying* donuts to shift governance polls led to the community deciding to make only locked (ie. untradeable) donuts eligible for use in voting. But there's an even cheaper attack than buying donuts (an attack that can be thought of as a kind of obfuscated bribe): *renting* them. If an attacker is already holding ETH, they can use it as collateral on a platform like [Compound](#) to take out a loan of some token, giving you the full right to use that token for whatever purpose including participating in votes, and when they're done they simply send the tokens back to the loan contract to get their collateral back - all without having to endure even a second of price exposure to the token that they just used to swing a coin vote, even if the coin vote mechanism includes a time lockup (as eg. Bihu does). In every case, issues around bribing, and accidentally over-empowering well-connected and wealthy participants, prove surprisingly difficult to avoid.

Identity

Some systems attempt to mitigate the plutocratic aspects of coin voting by making use of an identity system. In the case of the /r/ethtrader donut system, for example, although *governance polls* are done via coin vote, the mechanism that determines *how many donuts (ie. coins) you get in the first place* is based on Reddit accounts: 1 upvote from 1 Reddit account = N donuts earned. The ideal goal of an identity system is to

make it relatively easy for individuals to get one identity, but relatively difficult to get many identities. In the /r/ethtrader donut system, that's Reddit accounts, in the Gitcoin CLR matching gadget, it's Github accounts that are used for the same purpose. But identity, at least the way it has been implemented so far, is a fragile thing....

Oh, are you too lazy to make a big rack of phones? Well maybe you're looking for this:

Usual warning about how sketchy sites may or may not scam you, do your own research, etc. etc. applies.

Arguably, attacking these mechanisms by simply controlling thousands of fake identities like a puppetmaster is *even easier* than having to go through the trouble of bribing people. And if you think the response is to just increase security to go up to *government-level* IDs? Well, if you want to get a few of those you can start exploring here, but keep in mind that there are specialized criminal organizations that are well ahead of you, and even if all the underground ones are taken down, hostile governments are definitely going to create fake passports by the millions if we're stupid enough to create systems that make that sort of activity profitable. And this doesn't even begin to mention attacks in the opposite direction, identity-issuing institutions

attempting to disempower marginalized communities by *denying* them identity documents...

Collusion

Given that so many mechanisms seem to fail in such similar ways once multiple identities or even liquid markets get into the picture, one might ask, is there some deep common strand that causes all of these issues? I would argue the answer is yes, and the "common strand" is this: it is much harder, and more likely to be outright impossible, to make mechanisms that maintain desirable properties in a model where participants can collude, than in a model where they can't. Most people likely already have some intuition about this; specific instances of this principle are behind well-established norms and often laws promoting competitive markets and restricting price-fixing cartels, vote buying and selling, and bribery. But the issue is much deeper and more general.

In the version of game theory that focuses on individual choice - that is, the version that assumes that each participant makes decisions independently and that does not allow for the possibility of groups of agents working as one for their mutual benefit, there are [mathematical proofs](#) that at least one stable Nash equilibrium must exist in any game, and mechanism designers have a very wide latitude to "engineer" games to achieve specific outcomes. But in the version of game theory that allows for the possibility of coalitions working together, called *cooperative game theory*, **there are [large classes of games](#) that do not have any stable outcome that a coalition cannot profitably deviate from**.

Majority games, formally described as games of \(N\) agents where any subset of more than half of them can capture a fixed reward and split it among themselves, a setup eerily similar to many situations in corporate governance, politics and many other situations in human life, are [part of that set of inherently unstable games](#). That is to say, if there is a situation with some fixed pool of resources and some currently established mechanism for distributing those resources, and it's unavoidably possible for 51% of the participants can conspire to seize control of the resources, no matter what the current configuration is there is always some conspiracy that can emerge that would be profitable for the participants. However, that conspiracy would then in turn be vulnerable to potential new conspiracies, possibly including a combination of previous conspirators and victims... and so on and so forth.

Round	A	B	C
1	1/3	1/3	1/3
2	1/2	1/2	0
3	2/3	0	1/3
4	0	1/3	2/3

This fact, the instability of majority games under cooperative game theory, is arguably highly underrated as a simplified general mathematical model of why there may well be no "end of history" in politics and no system that proves fully satisfactory; I personally believe it's much more useful than the more famous [Arrow's theorem](#), for example.

There are two ways to get around this issue. The first is to try to restrict ourselves to the class of games that *are* "identity-free" and "collusion-safe", so where we do not need to worry about either bribes or identities. The second is to try to attack the identity and collusion resistance problems directly, and actually solve them well enough that we can implement non-collusion-safe games with the richer properties that they offer.

Identity-free and collusion-safe game design

The class of games that is identity-free and collusion-safe is substantial. Even proof of work is collusion-safe up to the bound of a single actor having [~23.21% of total hashpower](#), and this bound can be increased up to 50% with [clever engineering](#). Competitive markets are reasonably collusion-safe up until a relatively high bound, which is easily reached in some cases but in other cases is not.

In the case of *governance* and *content curation* (both of which are really just special cases of the general problem of identifying public goods and public bads) a major class of mechanism that works well is *[futarchy](#)* - typically portrayed as "governance by prediction market", though I would also argue that the use of security deposits is fundamentally in the same class of technique. The way futarchy mechanisms, in their most general form, work is that they make "voting" not just an expression of opinion, but also a *prediction*, with a reward for making predictions that are true and a penalty for making predictions that are false. For example, [my proposal](#) for "prediction markets for content curation DAOs" suggests a semi-centralized design where anyone can upvote or downvote submitted content, with content that is upvoted more being more visible, where there is also a "moderation panel" that makes final decisions. For each post, there is a small probability (proportional to the total volume of upvotes+downvotes on that post) that the moderation panel will be called on to make a final decision on the post. If the moderation panel approves a post, everyone who upvoted it is rewarded and everyone who downvoted it is penalized, and if the moderation panel disapproves a post the reverse happens; this mechanism encourages participants to make upvotes and downvotes that try to "predict" the moderation panel's judgements.

Another possible example of futarchy is a governance system for a project with a token, where anyone who votes for a decision is obligated to purchase some quantity of tokens at the price at the time the vote begins if the vote wins; this ensures that voting on a bad decision is costly, and in the limit if a bad decision wins a vote everyone who approved the decision must essentially buy out everyone else in the project. This ensures that an individual vote for a "wrong" decision can be very costly for the voter, precluding the possibility of cheap bribe attacks.
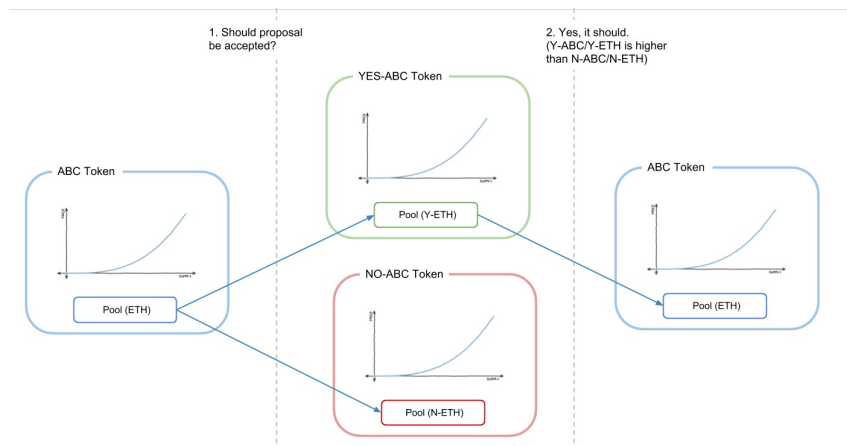
A graphical description of one form of futarchy, creating two markets representing the two "possible future worlds" and picking the one with a more favorable price. Source [this post on ethresear.ch](#)

However, that range of things that mechanisms of this type can do is limited. In the case of the content curation example above, we're not really solving governance, we're just *scaling* the functionality of a governance gadget that is already assumed to be trusted. One could try to replace the moderation panel with a prediction market on the price of a token representing the right to purchase advertising space, but in practice prices are too noisy an indicator to make this viable for anything but a very small number of very large decisions. And often the value that we're trying to maximize is explicitly something other than maximum value of a coin.

Let's take a more explicit look at why, in the more general case where we can't easily determine the value of a governance decision via its impact on the price of a token, good mechanisms for identifying public goods and bads unfortunately cannot be identity-free or collusion-safe. If one tries to preserve the property of a game being identity-free, building a system where identities don't matter and only coins do, **there is an impossible tradeoff between either failing to incentivize legitimate public goods or over-subsidizing plutocracy**.

The argument is as follows. Suppose that there is some author that is producing a public good (eg. a series of blog posts) that provides value to each member of a community of 10000 people. Suppose there exists some mechanism where members of the community can take an action that causes the author to receive a gain of $1. Unless the community members are *extremely* altruistic, for the mechanism to work the cost of taking this action must be much lower than $1, as otherwise the portion of the benefit captured by the member of the community supporting the author would be much smaller than the cost of supporting the author, and so the system collapses into a [tragedy of the commons](#) where no one supports the author. Hence, there must exist a way to cause the author to earn $1 at a cost much less than $1. But now suppose that there is also a fake community, which consists of 10000 fake sockpuppet accounts of the same wealthy attacker. This community takes all of the same actions as the real community, except instead of supporting the author, they support *another* fake account which is also a sockpuppet of the attacker. If it was possible for a member of the "real community" to give the author $1 at a personal cost of much less than $1, it's possible for the attacker to give *themselves* $1 at a cost much less than $1 over and over again, and thereby drain the system's funding. Any mechanism that can help genuinely under-coordinated parties coordinate will, without the right safeguards, also help already coordinated parties (such as many accounts controlled by the same person) *over-coordinate*, extracting money from the system.

A similar challenge arises when the goal is not funding, but rather determining what content should be most visible. What content do you think would get more dollar value supporting it: a legitimately high quality blog article benefiting thousands of people but benefiting each individual person relatively slightly, or this?

Or perhaps this?

Those who have been following recent politics "in the real world" might also point out a different kind of content that benefits highly centralized actors: social media manipulation by hostile governments. Ultimately, both centralized systems and decentralized systems are facing the same fundamental problem, which is that **the "marketplace of ideas" (and of public goods more generally) is very far from an "efficient market" in the sense that economists normally use the term**, and this leads to both underproduction of public goods even in "peacetime" but also vulnerability to active attacks. It's just a hard problem.

This is also why coin-based voting systems (like Bihu's) have one major genuine advantage over identity-based systems (like the Gitcoin CLR or the /r/ethtrader donut experiment): at least there is no benefit to buying accounts en masse, because everything you do is proportional to how many coins you have, regardless of how many accounts the coins are split between. However, mechanisms that do not rely on any model of identity and only rely on coins fundamentally cannot solve the problem of concentrated interests outcompeting dispersed communities trying to support public goods; an identity-free mechanism that empowers distributed communities cannot avoid over-empowering centralized plutocrats pretending to be distributed communities.

But it's not just identity issues that public goods games are vulnerable too; it's also bribes. To see why, consider again the example above, but where instead of the "fake community" being 10001 sockpuppets of the attacker, the attacker only has one identity, the account receiving funding, and the other 10000 accounts are real users - but users that receive a bribe of $0.01 each to take the action that would cause the attacker to gain an additional $1. As mentioned above, these bribes can be highly obfuscated, even through third-party custodial services that vote on a user's behalf in exchange for convenience, and in the case of "coin vote" designs an obfuscated bribe is even easier: one can do it by renting coins on the market and using them to participate in votes. Hence, while some kinds of games, particularly prediction market or security deposit based games, can be made collusion-safe and identity-free, generalized public goods funding seems to be a class of problem where collusion-safe and identity-free approaches unfortunately just cannot be made to work.

Collusion resistance and identity

The other alternative is attacking the identity problem head-on. As mentioned above, simply going up to higher-security centralized identity systems, like passports and other government IDs, will not work at scale; in a sufficiently incentivized context, they are very insecure and vulnerable to the issuing governments themselves! Rather, the kind of "identity" we are talking about here is some kind of robust multifactorial set of claims that an actor identified by some set of messages actually is a unique individual. A very early proto-model of this kind of networked identity is arguably social recovery in HTC's blockchain phone:
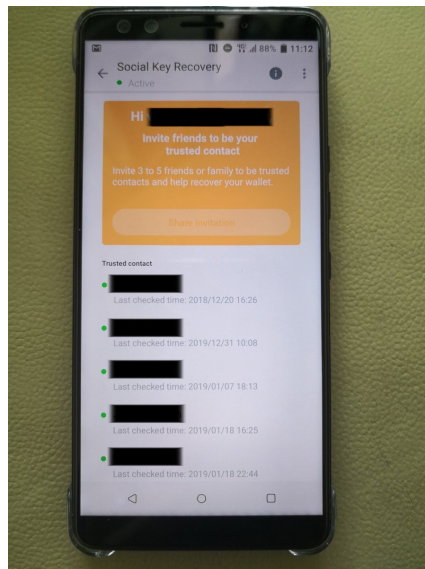
The basic idea is that your private key is secret-shared between up to five trusted contacts, in such a way that mathematically ensures that three of them can recover the original key, but two or fewer can't. This qualifies as an "identity system" - it's your five friends determining whether or not someone trying to recover your account actually is you. However, it's a special-purpose identity system trying to solve a problem - personal account security - that is different from (and easier than!) the problem of attempting to identify unique humans. That said, the general model of individuals making claims about each other can quite possibly be bootstrapped into some kind of more robust identity model. These systems could be augmented if desired using the "futarchy" mechanic described above: if someone makes a claim that someone is a unique human, and someone else disagrees, and both sides are willing to put down a bond to litigate the issue, the system can call together a judgement panel to determine who is right.

But we also want another crucially important property: we want an identity that you cannot credibly rent or sell. Obviously, we can't prevent people from making a deal "you send me $50, I'll send you my key", but what we *can* try to do is prevent such deals from being *credible* - make it so that the seller can easily cheat the buyer and give the buyer a key that doesn't actually work. One way to do this is to make a mechanism by which the owner of a key can send a transaction that revokes the key and replaces it with another key of the owner's choice, all in a way that cannot be proven. Perhaps the simplest way to get around this is to either use a trusted party that runs the computation and only publishes results (along with zero knowledge proofs proving the results, so the trusted party is trusted only for privacy, not integrity), or decentralize the same functionality through multi-party computation. Such approaches will not solve collusion completely; a group of friends could still come together and sit on the same couch and coordinate votes, but they will at least reduce it to a manageable extent that will not lead to these systems outright failing.

There is a further problem: initial distribution of the key. What happens if a user creates their identity inside a third-party custodial service that then stores the private key and uses it to clandestinely make votes on things? This would be an implicit bribe, the user's voting power in exchange for providing to the user a convenient service, and what's more, if the system is secure in that it successfully prevents bribes by making votes unprovable, clandestine voting by third-party hosts would *also* be undetectable. The only approach that gets around this problem seems to be.... in-person verification. For example, one could have an ecosystem of "issuers" where each issuer issues smart cards with private keys, which the user can immediately download onto their smartphone and send a message to replace the key with a different key that they do not reveal to anyone. These issuers could be meetups and conferences, or potentially individuals that have already been deemed by some voting mechanic to be trustworthy.

Building out the infrastructure for making collusion-resistant mechanisms possible, including robust decentralized identity systems, is a difficult challenge, but if we want to unlock the potential of such mechanisms, it seems unavoidable that we have to do our best to try. It is true that the current computer-security dogma around, for example, introducing online voting is simply "don't", but if we want to expand the role of voting-like mechanisms, including more advanced forms such as quadratic voting and quadratic finance, to more roles, we have no choice but to confront the challenge head-on, try really hard, and hopefully succeed at making something secure enough, for at least some use cases.

[Mirror] Cantor was Wrong: debunking the infinite set hierarchy

2019 Apr 01

This is a mirror of the post at https://medium.com/@VitalikButerin/cantor-was-wrong-debunking-the-infinite-set-hierarchy-e9ba5015102.

By Vitalik Buterin, PhD at University of Basel

A common strand of mathematics argues that, rather than being one single kind of infinity, there are actually an infinite hierarchy of different levels of infinity. Whereas the size of the set of integers is just plain infinite, and the set of rational numbers is just as big as the integers (because you can map every rational number to an integer by interleaving the digits of its numerator and denominator, eg. $0.456456456.... = \frac{456}{999} = \frac{152}{333} \rightarrow 135323$), the size of the set of *real* numbers is some kind of even bigger infinity, because there *is no way* to make a similar mapping from real numbers to the integers.

First of all, I should note that it's relatively easy to see that the claim that there is no mapping is false. Here's a simple mapping. For a given real number, give me a (deterministic) python program that will print out digits of it (eg. for π, that might be a program that calculates better and better approximations using the infinite series $\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + ...$). I can convert the program into a number (using `n = int.from_bytes(open('program.py').read(), 'big')`) and then output the number. Done. There's the mapping from real numbers to integers.

Now let's take a look at the most common argument used to claim that no such mapping can exist, namely Cantor's diagonal argument. Here's an [exposition from UC Denver](#); it's short so I'll just screenshot the whole thing:

Math 3000
Cantor's Diagonal Argument

Theorem

The set of real numbers $(0,1)$ is uncountable.

Proof

Assume that the real numbers in the set $(0,1)$ are countable. This means that these real numbers can be written in order as $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots,\}$. Each of these numbers can be written using a binary expansion. For notation,

$$x^{(k)} = 0.x_1^{(k)} x_2^{(k)} x_3^{(k)} x_4^{(k)} \cdots \quad \text{where } x_j^{(k)} \in \{0,1\}$$

Since the real numbers in $(0,1)$ are countable we can write

$$
\begin{array}{ccccccc}
x^{(1)} = & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & \cdots \\
x^{(2)} = & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \cdots \\
x^{(3)} = & x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & x_4^{(3)} & \cdots \\
x^{(4)} = & x_1^{(4)} & x_2^{(4)} & x_3^{(4)} & x_4^{(4)} & \cdots \\
\end{array}
$$

Now construct the number y where $y = 0.y_1 y_2 y_3 y_4 \cdots$. Choose $y_1 \neq x_1^{(1)}$, $y_2 \neq x_2^{(2)}, \ldots, y_j \neq x_j^{(j)}, \ldots$. This number is NOT in the original list $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots,\}$. This means that we have assumed that we have a countable list, and we have constructed a number that is not in the list. Therefore, we arrive at a contradiction that the real numbers are countable. QED

An example of such a construction is:

$$
\begin{array}{ccccccccccccc}
x^{(1)} = & \boxed{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\
x^{(2)} = & 0 & \boxed{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
x^{(3)} = & 1 & 0 & \boxed{1} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \cdots \\
x^{(4)} = & 0 & 1 & 0 & \boxed{1} & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\
x^{(5)} = & 0 & 1 & 1 & 0 & \boxed{1} & 1 & 0 & 1 & 1 & 0 & \cdots \\
x^{(6)} = & 1 & 0 & 1 & 0 & 0 & \boxed{1} & 0 & 1 & 0 & 1 & \cdots \\
x^{(7)} = & 1 & 0 & 1 & 0 & 0 & 1 & \boxed{0} & 0 & 1 & 0 & \cdots \\
x^{(8)} = & 0 & 1 & 1 & 0 & 1 & 0 & 1 & \boxed{0} & 1 & 0 & \cdots \\
x^{(9)} = & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \boxed{0} & 1 & \cdots \\
x^{(10)} = & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \boxed{1} & \cdots \\
\\
x_M \neq & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots \\
\end{array}
$$

Now, here's the fundamental flaw in this argument: *decimal expansions of real numbers are not unique*. To provide a counterexample in the exact format that the "proof" requires, consider the set (numbers written in binary), with diagonal digits bolded:

- x[1] = 0.**0**00000...

- x[2] = 0.0**1**1111...

- x[3] = 0.00**1**111...

- x[4] = 0.000**1**11...

- .....

The diagonal gives: 01111..... If we flip every digit, we get the number: \(y =\) 0.10000......

And here lies the problem: just as in decimal, [0.9999.... equals 1](#), in binary 0.01111..... equals 0.10000..... And so even though the new *decimal expansion* is not in the original list, the *number* \(y\)

is exactly the same as the number $x[2]$.

Note that this directly implies that the halting problem is in fact solvable. To see why, imagine a computer program that someone claims will not halt. Let c[1] be the state of the program after one step, c[2] after two steps, etc. Let x[1], x[2], x[3].... be a full enumeration of all real numbers (which exists, as we proved above), expressed in base 2^D where D is the size of the program's memory, so a program state can always be represented as a single "digit". Let y = 0.c[1]c[2]c[3]........ This number is by assumption part of the list, so it is one of the x[i] values, and hence it can be computed in some finite amount of time. This has implications in a number of industries, particularly in proving that "Turing-complete" blockchains are in fact secure.

Patent on this research is pending.

A CBC Casper Tutorial

2018 Dec 05

Special thanks to Vlad Zamfir, Aditya Asgaonkar, Ameen Soleimani and Jinglan Wang for review

In order to help more people understand "the other Casper" (Vlad Zamfir's CBC Casper), and specifically the instantiation that works best for blockchain protocols, I thought that I would write an explainer on it myself, from a less abstract and more "close to concrete usage" point of view. Vlad's descriptions of CBC Casper can be found [here](#) and [here](#) and [here](#); you are welcome and encouraged to look through these materials as well.

CBC Casper is designed to be fundamentally very versatile and abstract, and come to consensus on pretty much any data structure; you can use CBC to decide whether to choose 0 or 1, you can make a simple block-by-block chain run on top of CBC, or a 2^{92}-dimensional hypercube tangle DAG, and pretty much anything in between.

But for simplicity, we will first focus our attention on one concrete case: a simple chain-based structure. We will suppose that there is a fixed validator set consisting of N validators (a fancy word for "staking nodes"; we also assume that each node is staking the same amount of coins, cases where this is not true can be simulated by assigning some nodes multiple validator IDs), time is broken up into ten-second slots, and validator k can create a block in slot k, $N + k$, $2N + k$, etc. Each block points to one specific parent block. Clearly, if we wanted to make something maximally simple, we could just take this structure, impose a longest chain rule on top of it, and call it a day.
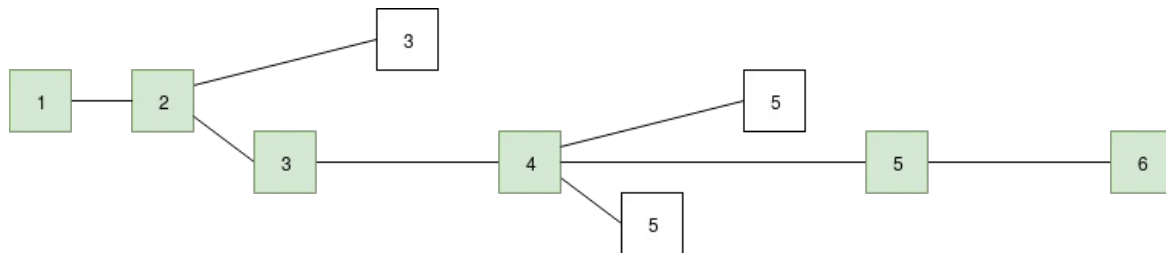
The green chain is the longest chain (length 6) so it is considered to be the "canonical chain".

However, what we care about here is adding some notion of "finality" - the idea that some block can be so firmly established in the chain that it cannot be overtaken by a competing block unless a very large portion (eg. $\frac{1}{4}$) of validators commit a *uniquely attributable fault* - act in some way which is clearly and cryptographically verifiably malicious. If a very large portion of validators *do* act maliciously to revert the block, proof of the misbehavior can be submitted to the chain to take away those validators' entire deposits, making the reversion of finality extremely expensive (think hundreds of millions of dollars).

LMD GHOST

We will take this one step at a time. First, we replace the fork choice rule (the rule that chooses which chain among many possible choices is "the canonical chain", ie. the chain that users should care about), moving away from the simple longest-chain-rule and instead using "latest message driven GHOST". To show how LMD GHOST works, we will modify the above example. To make it more concrete, suppose the validator set has size 5, which we label A, B, C, D, E, so validator A makes the blocks at slots 0 and 5, validator B at slots 1 and 6, etc. A client evaluating the LMD GHOST fork choice rule cares only about the most recent (ie. highest-slot) message (ie. block) signed by each validator:
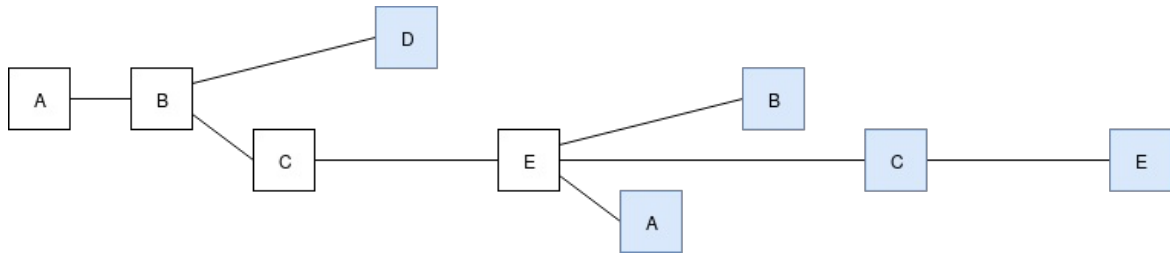
Latest messages in blue, slots from left to right (eg. \(A\)'s block on the left is at slot 0, etc.)

Now, we will use only these messages as source data for the "greedy heaviest observed subtree" (GHOST) fork choice rule: start at the genesis block, then each time there is a fork choose the side where more of the latest messages support that block's subtree (ie. more of the latest messages support either that block or one of its descendants), and keep doing this until you reach a block with no children. We can compute for each block the subset of latest messages that support either the block or one of its descendants:
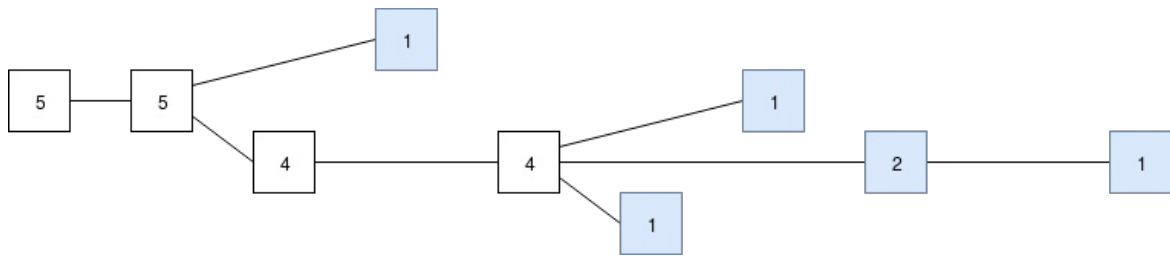
Now, to compute the head, we start at the beginning, and then at each fork pick the higher number: first, pick the bottom chain as it has 4 latest messages supporting it versus 1 for the single-block top chain, then at the next fork support the middle chain. The result is the same longest chain as before. Indeed, in a well-running network (ie. the orphan rate is low), almost all of the time LMD GHOST and the longest chain rule *will* give the exact same answer. But in more extreme circumstances, this is not always true. For example, consider the following chain, with a more substantial three-block fork:
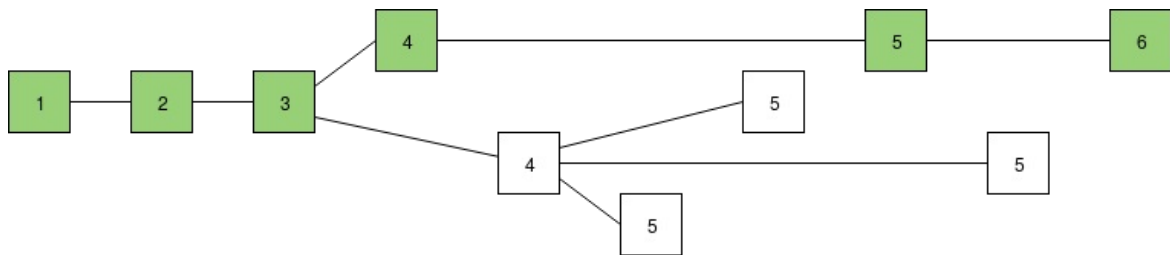
Scoring blocks by chain length. If we follow the longest chain rule, the top chain is longer, so the top chain wins.
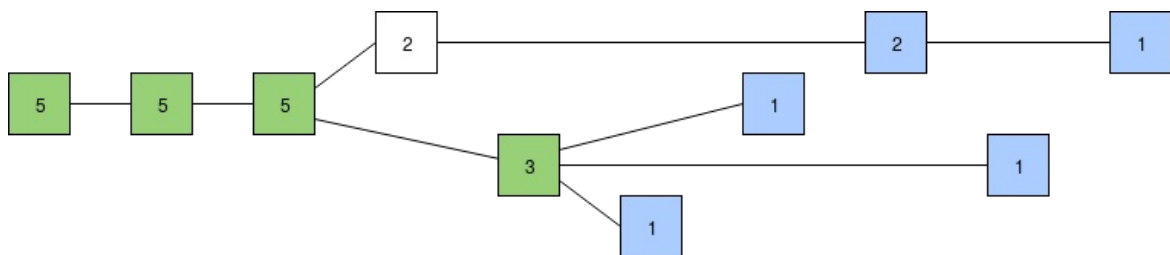
Scoring blocks by number of supporting latest messages and using the GHOST rule (latest message from each validator shown in blue). The bottom chain has more recent support, so if we follow the LMD GHOST rule the bottom chain wins, though it's not yet clear which of the three blocks takes precedence.

The LMD GHOST approach is advantageous in part because it is better at extracting information in conditions of high latency. If two validators create two blocks with the same parent, they should really be both counted as cooperating votes for the parent block, even though they are at the same time competing votes for themselves. The longest chain rule fails to capture this nuance; GHOST-based rules do.

Detecting finality

But the LMD GHOST approach has another nice property: it's *sticky*. For example, suppose that for two rounds, $\frac{4}{5}$ of validators voted for the same chain (we'll assume that the one of the five validators that did not, B, is attacking):
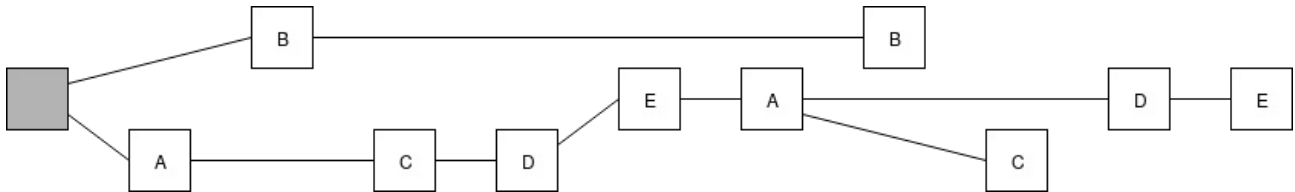
What would need to actually happen for the chain on top to become the canonical chain? Four of five validators built on top of E's first block, and all four recognized that E had a high score in the LMD fork choice. Just by looking at the structure of the chain, we can know for a fact at least some of the messages that the validators must have seen at different times. Here is what we know about the four validators' views:
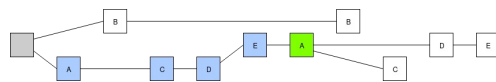
A's view
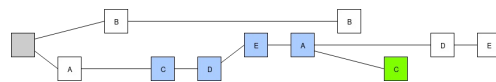
C's view
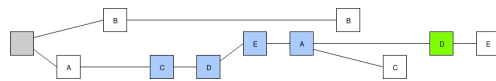
D's view
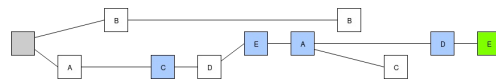
E's view

Blocks produced by each validator in green, the latest messages we know that they saw from each of the other validators in blue.

Note that all four of the validators *could have* seen one or both of B's blocks, and D and E *could have* seen C's second block, making that the latest message in their views instead of C's first block; however, the structure of the chain itself gives us no evidence that they actually did. Fortunately, as we will see below, this ambiguity does not matter for us.

A's view contains four latest-messages supporting the bottom chain, and none supporting B's block. Hence, in (our simulation of) A's eyes the score in favor of the bottom chain is *at least* 4-1. The views of C, D and E paint a similar picture, with four latest-messages supporting the bottom chain. Hence, all four of the validators are in a position where they cannot change their minds unless two other validators change their minds first to bring the score to 2-3 in favor of B's block.

Note that our simulation of the validators' views is "out of date" in that, for example, it does not capture that D and E could have seen the more recent block by C. However, this does not alter the calculation for the top vs bottom chain, because we can very generally say that any validator's new message will have the same opinion as their previous messages, unless two other validators have already switched sides first.
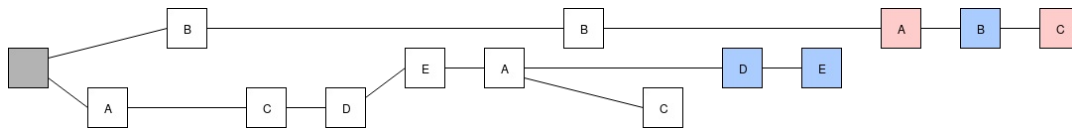
A minimal viable attack. A and C illegally switch over to support B's block (and can get penalized for this), giving it a 3-2 advantage, and at this point it becomes legal for D and E to also switch over.

Since fork choice rules such as LMD GHOST are sticky in this way, and clients can detect when the fork choice rule is "stuck on" a particular block, we can use this as a way of achieving asynchronously safe consensus.

Safety Oracles

Actually detecting all possible situations where the chain becomes stuck on some block (in CBC lingo, the block is "decided" or "safe") is very difficult, but we can come up with a set of heuristics ("safety oracles") which will help us detect *some* of the cases where this happens. The simplest of these is the **clique oracle**. If there exists some subset V of the validators making up portion p of the total validator set (with $p > \frac{1}{2}$) that all make blocks supporting some block B and then make another round of blocks still supporting B that references their first round of blocks, then we can reason as follows:

Because of the two rounds of messaging, we know that this subset V all (i) support B (ii) know that B is well-supported, and so none of them can legally switch over unless enough others switch over first. For some competing B' to beat out B, the support such a B' can *legally* have is initially at most $(1-p)$ (everyone not part of the clique), and to win the LMD GHOST fork choice its support needs to get to $\frac{1}{2}$, so at least $\frac{1}{2} - (1-p) = p - \frac{1}{2}$ need to illegally switch over to get it to the point where the LMD GHOST rule supports B'.

As a specific case, note that the $p=\frac{3}{4}$ clique oracle offers a $\frac{1}{4}$ level of safety, and a set of blocks satisfying the clique can (and in normal operation, will) be generated as long as $\frac{3}{4}$ of nodes are online. Hence, in a BFT sense, the level of fault tolerance that can be reached using two-round clique oracles is $\frac{1}{3}$, in terms of both liveness and safety.

This approach to consensus has many nice benefits. First of all, the short-term chain selection algorithm, and the "finality algorithm", are not two awkwardly glued together distinct components, as they admittedly are in Casper FFG; rather, they are both part of the same coherent whole. Second, because safety detection is client-side, there is no need to choose any thresholds in-protocol; clients can decide for themselves what level of safety is sufficient to consider a block as finalized.

Going Further

CBC can be extended further in many ways. First, one can come up with other safety oracles; higher-round clique oracles can reach $\frac{1}{3}$ fault tolerance. Second, we can add validator rotation mechanisms. The simplest is to allow the validator set to change by a small percentage every time the $q=\frac{3}{4}$ clique oracle is satisfied, but there are other things that we can do as well. Third, we can go beyond chain-like structures, and instead look at structures that increase the density of messages per unit time, like the Serenity beacon chain's attestation structure:
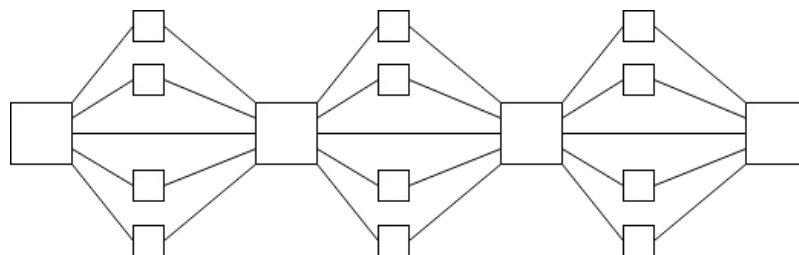
In this case, it becomes worthwhile to separate *attestations* from *blocks*; a block is an object that

actually grows the underlying DAG, whereas an attestation contributes to the fork choice rule. In the Serenity beacon chain spec, each block may have hundreds of attestations corresponding to it. However, regardless of which way you do it, the core logic of CBC Casper remains the same.

To make CBC Casper's safety "cryptoeconomically enforceable", we need to add validity and slashing conditions. First, we'll start with the validity rule. A block contains both a parent block and a set of attestations that it knows about that are not yet part of the chain (similar to "uncles" in the current Ethereum PoW chain). For the block to be valid, the block's parent must be the result of executing the LMD GHOST fork choice rule given the information included in the chain including in the block itself.
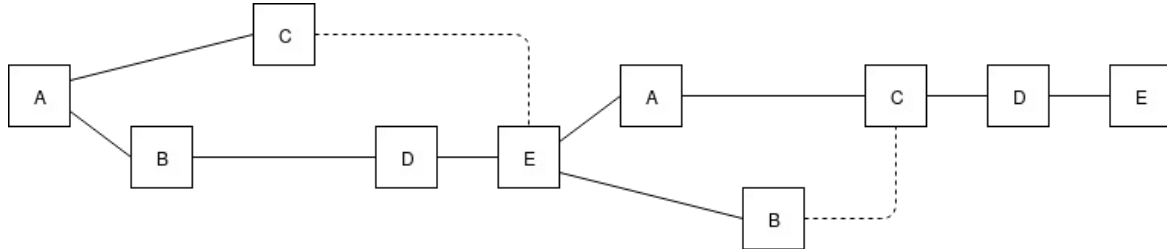
Dotted lines are uncle links, eg. when E creates a block, E notices that C is not yet part of the chain, and so includes a reference to C.

We now can make CBC Casper safe with only one slashing condition: you cannot make two attestations M_1 and M_2, unless either M_1 is in the chain that M_2 is attesting to or M_2 is in the chain that M_2 is attesting to.

OK   Not OK
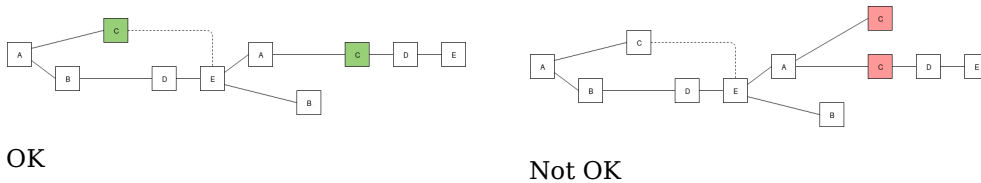
The validity and slashing conditions are relatively easy to describe, though actually implementing them requires checking hash chains and executing fork choice rules in-consensus, so it is not nearly as simple as taking two messages and checking a couple of inequalities between the numbers that these messages commit to, as you can do in Casper FFG for the `NO_SURROUND` and `NO_DBL_VOTE` slashing conditions.

Liveness in CBC Casper piggybacks off of the liveness of whatever the underlying chain algorithm is (eg. if it's one-block-per-slot, then it depends on a synchrony assumption that all nodes will see everything produced in slot N before the start of slot $N+1$). It's not possible to get "stuck" in such a way that one cannot make progress; it's possible to get to the point of finalizing new blocks from any situation, even one where there are attackers and/or network latency is higher than that required by the underlying chain algorithm.

Suppose that at some time T, the network "calms down" and synchrony assumptions are once again satisfied. Then, everyone will converge on the same view of the chain, with the same head H. From there, validators will begin to sign messages supporting H or descendants of H. From there, the chain can proceed smoothly, and will eventually satisfy a clique oracle, at which point H becomes finalized.
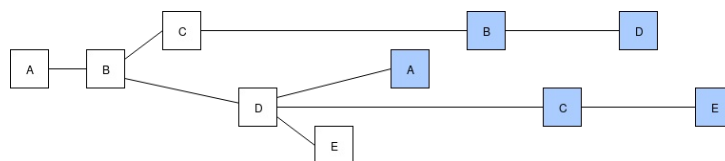
Chaotic network due to high latency.
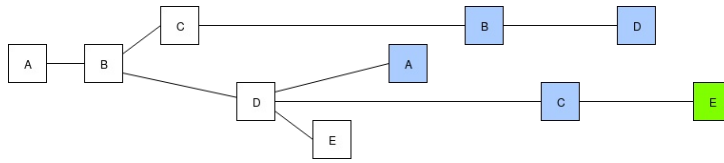
Network latency subsides, a majority of validators see all of the same blocks or at least enough of them to get to the same head when executing the fork choice, and start building on the head, further reinforcing its advantage in the fork choice rule.
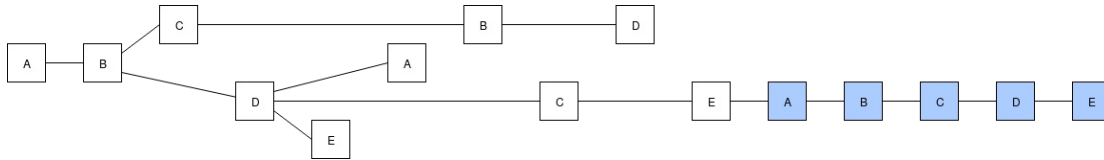
Chain proceeds "peacefully" at low latency. Soon, a clique oracle will be satisfied.

That's all there is to it! Implementation-wise, CBC may arguably be considerably more complex than FFG, but in terms of ability to reason about the protocol, and the properties that it provides, it's surprisingly simple.

[Mirror] Central Planning as Overfitting

This is a mirror of the post at https://radicalxchange.org/blog/posts/2018-11-26-4m9b8b/ written by myself and Glen Weyl

There is an intuition shared by many that "central planning" — command-and-control techniques for allocating resources in economies, and fine-grained knob-turning interventionism more generally — is undesirable. There's quite a lot to this, but it is often misapplied in a way that also leads it to go under-appreciated. In this post we try to clarify the appropriate scope of the intuition.

Some recent examples of the intuition being misapplied are:

- People arguing that relatively simple entitlement programs like Social Security are burdensome government intervention, while elaborate and often discretionary tax breaks conditional on specific behaviors are a good step towards less government.
- People arguing that block size limits in cryptocurrencies, which impose a hard cap on the number of transactions that each block can contain, are a form of central planning, but who do not argue against other centrally planned parameters, eg. the targeted ten minute (or whatever) average time interval between blocks.
- People arguing that a lack of block size limits constitutes central planning (!!)
- People arguing that fixed transaction fees constitute central planning, but variable transaction fees that arise out of an equilibrium itself created by a fixed block size limit do not.
- We were recently at a discussion in policy circles in Washington, where one of us was arguing for a scheme based on Harberger taxes for spectrum licenses, debating against someone defending more conventional perpetual monopoly licenses on spectrum aggregated at large levels that would tend to create a few national cellular carriers. The latter side argued that the Harberger tax scheme constituted unacceptable bureaucratic interventionism, but seemed to believe that permanent government-issued monopoly privileges are a property right as natural as apples falling from a tree.

While we do not entirely dismiss this last example, for reasons we will return to later, it does seem overplayed. Similarly and conversely, we see many examples where, in the name of defending or promoting "markets" (or at least "economic rationality") many professional economists advocate schemes that seem much more to us like central planning than the systems they seek to replace:

- The most systematic example of this is the literature on "optimal mechanism design," which began with the already extremely complicated and fragile Vickrey-Clarke-Groves mechanism and has only gotten more byzantine from there. While Vickrey's motivation for these ideas was to discover relatively simple rules that would correct the flaws of standard capitalism, he acknowledged in his paper that the design was highly complex in its direct application and urged future researchers to find simplifications. Instead of following this counsel, many scholars have proposed, for example, schemes that rely on a central authority being able to specify an infinite dimensional set of prior beliefs. These schemes, we submit, constitute "central planning" in precisely the sense we should be concerned with.
- Furthermore, these designs are not just matters of theory, but in practice many applied mechanism designers have created systems with similar properties. The recent United States Spectrum Incentive auctions (designed by a few prominent economists and computer scientists) centralized the enforcement of potential conflicts between transmission rights using an extremely elaborate and opaque computational engine, rather than allowing conflicts to be resolved through (for example) common law liability lawsuits as other interference between property claims and land uses are. A recent design for the allocation of courses to students at the University of Pennsylvania designed by a similar team requires students to express their preferences over courses on a novel numerical scale, allowing them only narrow language for expressing complementarities and substitutability between courses and then uses a state-of-the-art optimization engine to allocate courses. Auction systems designed by economists and computer scientists at large technology companies, like Facebook and Google, are even richer and less transparent, and have created substantial backlash, inspiring a whole industry of firms that help advertisers optimize their bidding in elaborate ways against these systems.
- This problem does not merely arise in mechanism design, however. In the fields of industrial organization (the basis of much antitrust economics) and the field of macroeconomics (the basis

of much monetary policy), extremely elaborate models with hundreds of parameters are empirically estimated and used to simulate the effects of mergers or changes in monetary policy. These models are usually difficult to explain even to experts in the field, much less democratically-elected politicians, judges or, god forbid, voters. And yet the confidence we have in these models, the empirical evidence validating their accuracy, etc. is almost nothing. Nonetheless, economists consistently promote such methods as "the state of the art" and they are generally viewed positively by defenders of the "market economy".

To understand why we think the concept of "intervention" is being misapplied here, we need to understand two different ways of measuring the extent to which some scheme is "interventionist". The first approach is to try to measure the absolute magnitude of distortion relative to some imagined state of nature (anarcho-primitivism, or a blockchain with no block size limits, or...). However, this approach clearly fails to capture the intuitions of why central planning is undesirable. For example, property rights in the physical world are a large intervention into almost every person's behavior, considerably limiting the actions that we can take every day. Many of these restrictions are actually of quite recent historical provenance (beginning with agriculture, and mostly in the West and not the East or Middle East). However, opponents of central planning often tend to be the strongest proponents of property rights!

We can shed some light on this puzzle by looking at another way of measuring the "central-planny-ness" of some social structure: in short, measure the number of knobs. Property rights actually score quite well here: every piece of property is allocated to some person or legal entity, they can use it as they wish, and no one else can touch it without their permission. There are choices to make around the edges (eg. adverse possession rights), but generally there isn't too much room for changing the scheme around (though note that privatization schemes, ie. transitions from something other than property rights to property rights like the auctions we discussed above, have very many knobs, and so there we can see more risks). Command-and-control regulations with ten thousand clauses (or market designs that specify elaborate probabilistic objects, or optimization protocols, etc.), or attempts to limit use of specific features of the blockchain to drive out specific categories of users, are much less desirable, as such strategies leave many more choices to central planners. A block size limit and a fixed transaction fee (or carbon taxes and a cap-and-trade scheme) have the exact same level of "central-planny-ness" to them: one variable (either quantity or price) is fixed in the protocol, and the other variable is left to the market.

Here are some key underlying reasons why we believe that simple social systems with fewer knobs are so desirable:

- They have **fewer moving parts that can fail** or otherwise have unexpected effects.
- They are **less likely to overfit**. If a social system is too complex, there are many parameters to set and relatively few experiences to draw from when setting the parameters, making it more likely that the parameters will be set to overfit to one context, and generalize poorly to other places or to future times. **We know little**, and we should not design systems that demand us to know a lot.
- They are **more resistant to corruption**. If a social system is simpler, it is more difficult (not impossible, but still more difficult) to set parameters in ways that encode attempts to privilege or anti-privilege specific individuals, factions or communities. This is not only good because it leads to fairness, it is also good because it leads to less zero-sum conflict.
- They can **more easily achieve legitimacy**. Because simpler systems are easier to understand, and easier for people to understand that a given implementation is not unfairly privileging special interests, it is easier to create common knowledge that the system is fair, creating a stronger sense of trust. Legitimacy is perhaps the central virtue of social institutions, as it sustains cooperation and coordination, enables the possibility of democracy (how can you democratically participate and endorse a system you do not understand?) and allows for a bottoms-up, rather than top-down, creation of a such a system, ensuring it can be implemented without much coercion or violence.

These effects are not always achieved (for example, even if a system has very few knobs, it's often the case that there exists a knob that can be turned to privilege well-connected and wealthy people as a class over everyone else), but the simpler a system is, the more likely the effects are to be achieved.

While avoiding over-complexity and overfit in personal decision-making is also important, avoiding these issues in large-scale social systems is even more important, because of the inevitable possibility of powerful forces attempting to manipulate knobs for the benefit of special interests, and the need to achieve common knowledge that the system has not been greatly corrupted, to the point where the fairness of the system is obvious even to unsophisticated observers.

This is not to condemn all forms or uses of complexity in social systems. Most science and the inner workings of many technical systems are likely to be opaque to the public but this does not mean science or technology is useless in social life; far from it. However, these systems, to gain legitimacy, usually show that they can reliably achieve some goal, which is transparent and verifiable. Planes land safely and on time, computational systems seem to deliver calculations that are correct, etc. It is by this process of verification, rather than by the transparency of the system per se, that such systems gain their legitimacy. However, for many social systems, truly large-scale, repeatable tests are difficult if not impossible. As such, simplicity is usually critical to legitimacy.

Different Notions of Simplicity

However, there is one class of social systems that seem to be desirable, and that intellectual advocates of minimizing central planning tend to agree are desirable, that don't quite fit the simple "few knobs" characterization that we made above. For example, consider common law. Common law is built up over thousands of precedents, and contains a large number of concepts (eg. see this list under "property law", itself only a part of common law; have you heard of "quicquid plantatur solo, solo cedit" before?). However, proponents of private property are very frequently proponents of common law. So what gives?

Here, we need to make a distinction between redundant complexity, or many knobs that really all serve a relatively small number of similar goals, and optimizing complexity, in the extreme one knob per problem that the system has encountered. In computational complexity theory, we typically talk about Kolmogorov complexity, but there are other notions of complexity that are also useful here, particularly VC dimension - roughly, the size of the largest set of situations for which we can turn the knobs in a particular way to achieve any particular set of outcomes. Many successful machine learning techniques, such as Support Vector Machines and Boosting, are quite complex, both in the formal Kolmogorov sense and in terms of the outcomes they produce, but can be proven to have low VC dimension.

VC dimension does a nice job capturing some of the arguments for simplicity mentioned above more explicitly, for example:

- A system with low VC dimension may have some moving parts that fail, but if it does, its different constituent parts can correct for each other. By construction, it has built in resilience through redundancy
- Low VC dimension is literally a measure of resistance to overfit.
- Low VC dimension leads to resistance to corruption, because if VC dimension is low, a corrupt or self-interested party in control of some knobs will not as easily be able to achieve some particular outcome that they desire. In particular, this agent will be "checked and balanced" by other parts of the system that redundantly achieve the originally desired ends.
- They can achieve legitimacy because people can randomly check a few parts and verify in detail that those parts work in ways that are reasonable, and assume that the rest of the system works in a similar way. An example of this was the ratification of the United States Constitution which, while quite elaborate, was primarily elaborate in the redundancy with which it applied the principle of checks and balances of power. Thus most citizens only read one or a few of The Federalist Papers that explained and defended the Constitution, and yet got a reasonable sense for what was going on.

This is not as clean and convenient as a system with low Kolmogorov complexity, but still much better than a system with high complexity where the complexity is "optimizing" (for an example of this in the blockchain context, see Vitalik's opposition and alternative to on-chain governance). The primary disadvantage we see in Kolmogorov complex but VC simple designs is for new social institutions, where it may be hard to persuade the public that these are VC simple. VC simplicity is usually easier as a basis for legitimacy when an institutions has clearly been built up without any clear design over a long period of time or by a large committee of people with conflicting interests (as with the United States Constitution). Thus when offering innovations we tend to focus more on Kolmogorov simplicity and hope many redundant each Kolmogorov-simple elements will add up to a VC-simple system. However, we may just not have the imagination to think of how VC simplicity might be effectively explained.

There are forms of the "avoid central planning" intuition that are misunderstandings and ultimately counterproductive. For example, try to automatically seize upon designs that seem at first glance to "look like a market", because not all markets are created equal. For example, one of us has argued for using fixed prices in certain settings to reduce uncertainty, and the other has (for similar information sharing reasons) argued for auctions that are a synthesis of standard descending price Dutch and ascending price English auctions (Channel auctions). That said, it is also equally a large error to throw the intuition away entirely. Rather, it is a valuable and important insight that can

easily is central to the methodology we have been recently trying to develop. Simplicity to Whom? Or Why Humanities Matter

However, the academic critics of this type of work are not simply confused. There is a reasonable basis for unease with discussions of "simplicity" because they inevitably contain a degree of subjectivity. What is "simple" to describe or appears to have few knobs in one language for describing it is devilishly complex in another, and vice versa. A few examples should help illuminate the point:

- We have repeatedly referred to "knobs", which are roughly real valued parameters. But real-valued parameters can encode an arbitrary amount of complexity. For example, I could claim my system has only one knob, it is just that slight changes in the 1000th decimal place of the setting of that knob end up determining incredibly important properties of the system. This may seem cheap, but more broadly it is the case that non-linear mappings between systems can make one system seem "simple" and another "complex" and in general there is just no way to say which is right.
- Many think of the electoral system of the United States as "simple", and yet, if one reflects on it or tries to explain it to a foreigner, it is almost impossible to describe. It is familiar, not simple, and we just have given a label to it ("the American system") that lets us refer to it in a few words. Systems like Quadratic Voting, or ranked choice voting, are often described as complex, but this seems to have more to do with lack of familiarity than complexity.
- Many scientific concepts, such as the "light cone", are the simplest thing possible once one understands special relativity and yet are utterly foreign and bizarre without having wrapped one's hands around this theory.

Even Kolmogorov complexity (length of the shortest computer program that encodes some given system) is relative to some programming language. Now, to some extent, VC dimension offers a solution: it says that a class of systems is simple if it is not too flexible. But consider what happens when you try to apply this; to do so, let's return to our example upfront about Harberger taxes v. perpetual licenses for spectrum.

Harberger taxes strike us as quite simple: there is a single tax rate (and the theory even says this is tied down by the rate at which assets turnover, at least if we want to maximally favor allocative efficiency) and the system can be described in a sentence or two. It seems pretty clear that such a system could not be contorted to achieve arbitrary ends. However, an opponent could claim that we chose the Harberger tax from an array of millions of possible mechanisms of a similar class to achieve a specific objective, and it just sounds simple (as with our examples of "deceptive" simplicity above).

To counter this argument, we would respond that the Harberger tax, or very similar ideas, have been repeatedly discovered or used (to some success) throughout human history, beginning with the Greeks, and that we do not propose this system simply for spectrum licenses but in a wide range of contexts. The chances that in all these contexts we are cherry-picking the system to "fit" that setting seems low. We would submit to the critic to judge whether it is really plausible that all these historical circumstances and these wide range of applications just "happen" to coincide.

Focusing on familiarity (ie. conservatism), rather than simplicity in some abstract mathematical sense, also carries many of the benefits of simplicity as we described above; after all, familiarity is simplicity, if the language we are using to describe ideas includes references to our shared historical experience. Familiar mechanisms also have the benefit that we have more knowledge of how similar ideas historically worked in practice. So why not just be conservative, and favor perpetual property licenses strongly over Harberger taxes?

There are three flaws in that logic, it seems to us. First, to the extent it is applied, it should be applied uniformly to all innovation, not merely to new social institutions. Technologies like the internet have contributed greatly to human progress, but have also led to significant social upheavals; this is not a reason to stop trying to advance our technologies and systems for communication, and it is not a reason to stop trying to advance our social technologies for allocating scarce resources.

Second, the benefits of innovation are real, and social institutions stand to benefit from growing human intellectual progress as much as everything else. The theoretical case for Harberger taxes providing efficiency benefits is strong, and there is great social value in doing small and medium-scale experiments to try ideas like them out. Investing in experiments today increases what we know, and so increases the scope of what can be done "conservatively" tomorrow.

Third, and most importantly, the cultural context in which you as a decision maker have grown up today is far from the only culture that has existed on earth. Even at present, Singapore, China,

Taiwan and Scandinavia have had significant success with quite different property regimes than the United States. Video game developers and internet protocol designers have had to solve incentive problems of a similar character to what we see today in the blockchain space and have come up with many kinds of solutions, and throughout history, we have seen a wide variety of social systems used for different purposes, with a wide range of resulting outcomes. By learning about the different ways in which societies have lived, understood what is natural and imagined their politics, we can gain the benefits of learning from historical experience and yet at the same time open ourselves to a much broader space of possible ideas to work with.

This is why we believe that balance and collaboration between different modes of learning and understanding, both the mathematical one of economists and computer scientists, and the historical experiences studied by historians, anthropologists, political scientists, etc is critical to avoid the mix of and often veering between extreme conservatism and dangerous utopianism that has become characteristic of much intellectual discourse in e.g. the economics community, the "rationalist" community, and in [many](#) [cases](#) blockchain protocol design.

Layer 1 Should Be Innovative in the Short Term but Less in the Long Term

2018 Aug 26

See update 2018-08-29

One of the key tradeoffs in blockchain design is whether to build more functionality into base-layer blockchains themselves ("layer 1"), or to build it into protocols that live on top of the blockchain, and can be created and modified without changing the blockchain itself ("layer 2"). The tradeoff has so far shown itself most in the scaling debates, with block size increases (and sharding) on one side and layer-2 solutions like Plasma and channels on the other, and to some extent blockchain governance, with loss and theft recovery being solvable by either the DAO fork or generalizations thereof such as EIP 867, or by layer-2 solutions such as Reversible Ether (RETH). So which approach is ultimately better? Those who know me well, or have seen me out myself as a dirty centrist, know that I will inevitably say "some of both". However, in the longer term, I do think that as blockchains become more and more mature, layer 1 will necessarily stabilize, and layer 2 will take on more and more of the burden of ongoing innovation and change.

There are several reasons why. The first is that layer 1 solutions require ongoing protocol change to happen at the base protocol layer, base layer protocol change requires governance, and **it has still not been shown that, in the long term, highly "activist" blockchain governance can continue without causing ongoing political uncertainty or collapsing into centralization**.

To take an example from another sphere, consider Moxie Marlinspike's defense of Signal's centralized and non-federated nature. A document by a company defending its right to maintain control over an ecosystem it depends on for its key business should of course be viewed with massive grains of salt, but one can still benefit from the arguments. Quoting:

> One of the controversial things we did with Signal early on was to build it as an unfederated service. Nothing about any of the protocols we've developed requires centralization; it's entirely possible to build a federated Signal Protocol-based messenger, but I no longer believe that it is possible to build a competitive federated messenger at all.

And:

> Their retort was "that's dumb, how far would the internet have gotten without interoperable protocols defined by 3rd parties?" I thought about it. We got to the first production version of IP, and have been trying for the past 20 years to switch to a second production version of IP with limited success. We got to HTTP version 1.1 in 1997, and have been stuck there until now. Likewise, SMTP, IRC, DNS, XMPP, are all similarly frozen in time circa the late 1990s. To answer his question, that's how far the internet got. It got to the late 90s. That has taken us pretty far, but it's undeniable that once you federate your protocol, it becomes very difficult to make changes. And right now, at the application level, things that stand still don't fare very well in a world where the ecosystem is moving ... So long as federation means stasis while centralization means movement, federated protocols are going to have trouble existing in a software climate that demands movement as it does today.

At this point in time, and in the medium term going forward, it seems clear that decentralized application platforms, cryptocurrency payments, identity systems, reputation systems, decentralized exchange mechanisms, auctions, privacy solutions, programming languages that support privacy solutions, and most other interesting things that can be done on blockchains are spheres where there will continue to be significant and ongoing innovation. Decentralized application platforms often need continued reductions in confirmation time, payments need fast confirmations, low transaction costs, privacy, and many other built-in features, exchanges are appearing in many shapes and sizes including on-chain automated market makers, frequent batch auctions, combinatorial auctions and more. Hence, "building in" any of these into a base layer blockchain would be a bad idea, as it would create a high level of governance overhead as the platform would have to continually discuss, implement and coordinate newly discovered technical improvements. For the same reason federated messengers have a hard time getting off the ground without re-centralizing, blockchains would also need to choose between adopting activist governance, with the perils that entails, and falling behind newly appearing alternatives.

Even Ethereum's limited level of application-specific functionality, precompiles, has seen some of this effect. Less than a year ago, Ethereum adopted the Byzantium hard fork, including operations to facilitate elliptic curve operations needed for ring signatures, ZK-SNARKs and other applications, using the alt-bn128 curve. Now, Zcash and other blockchains are moving toward BLS-12-381, and Ethereum would need to fork again to catch up. In part to avoid having similar problems in the future, the Ethereum community is looking to upgrade the EVM to E-WASM, a virtual machine that is sufficiently more efficient that there is far less need to incorporate application-specific precompiles.

But there is also a second argument in favor of layer 2 solutions, one that does not depend on speed of anticipated technical development: *sometimes there are inevitable tradeoffs, with no single globally optimal solution*. This is less easily visible in Ethereum 1.0-style blockchains, where there are certain models that are reasonably universal (eg. Ethereum's account-based model is one). In *sharded* blockchains, however, one type of question that does *not* exist in Ethereum today crops up: how to do cross-shard transactions? That is, suppose that the blockchain state has regions A and B, where few or no nodes are processing both A and B. How does the system handle transactions that affect both A and B?

The current answer involves asynchronous cross-shard communication, which is sufficient for transferring assets and some other applications, but insufficient for many others. Synchronous operations (eg. to solve the train and hotel problem) can be bolted on top with cross-shard yanking, but this requires multiple rounds of cross-shard interaction, leading to significant delays. We can solve these problems with a synchronous execution scheme, but this comes with several tradeoffs:

- The system cannot process more than one transaction for the same account per block
- Transactions must declare in advance what shards and addresses they affect
- There is a high risk of any given transaction failing (and still being required to pay fees!) if the transaction is only accepted in some of the shards that it affects but not others

It seems very likely that a better scheme can be developed, but it would be more complex, and may well have limitations that this scheme does not. There are known results preventing perfection; at the very least, Amdahl's law puts a hard limit on the ability of some applications and some types of interaction to process more transactions per second through parallelization.

So how do we create an environment where better schemes can be tested and deployed? The answer is an idea that can be credited to Justin Drake: layer 2 execution engines. Users would be able to send assets into a "bridge contract", which would calculate (using some indirect technique such as interactive verification or ZK-SNARKs) state roots using some alternative set of rules for processing the blockchain (think of this as equivalent to layer-two "meta-protocols" like Mastercoin/OMNI and Counterparty on top of Bitcoin, except because of the bridge contract these protocols would be able to handle assets whose "base ledger" is defined on the underlying protocol), and which would process withdrawals if and only if the alternative ruleset generates a withdrawal request.
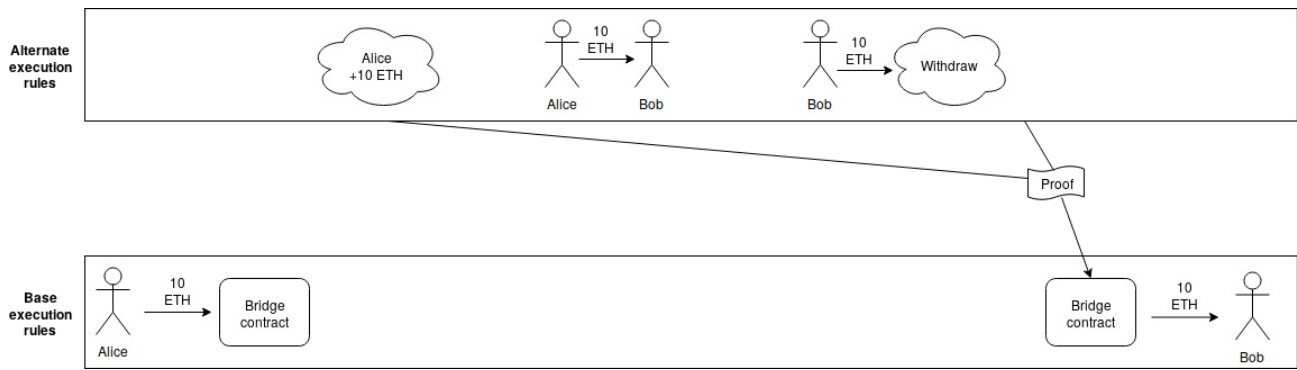
Note that anyone can create a layer 2 execution engine at any time, different users can use different execution engines, and one can switch from one execution engine to any other, or to the base protocol, fairly quickly. The base blockchain no longer has to worry about being an optimal smart contract processing engine; it need only be a data availability layer with execution rules that are quasi-Turing-complete so that any layer 2 bridge contract can be built on top, and that allow basic operations to carry state between shards (in fact, only ETH transfers being fungible across shards is sufficient, but it takes very little effort to also allow cross-shard calls, so we may as well support them), but does not require complexity beyond that. Note also that layer 2 execution engines can have different state management rules than layer 1, eg. not having storage rent; anything goes, as it's the responsibility of the users of that specific execution engine to make sure that it is sustainable, and if they fail to do so the consequences are contained to within the users of that particular execution engine.

In the long run, layer 1 would not be actively competing on all of these improvements; it would simply provide a stable platform for the layer 2 innovation to happen on top. **Does this mean that, say, sharding is a bad idea, and we should keep the blockchain size and state small so that even 10 year old computers can process everyone's transactions? Absolutely not.** Even if execution engines are something that gets partially or fully moved to layer 2, consensus on data ordering and availability is still a highly generalizable and necessary function; to see how difficult layer 2 execution engines are without layer 1 scalable data availability consensus, see the difficulties in Plasma research, and its difficulty of naturally extending to fully general purpose blockchains, for an example. And if people want to throw a hundred megabytes per second of data into a system where they need consensus on availability, then we need a hundred megabytes per second of data availability consensus.

Additionally, layer 1 can still improve on reducing latency; if layer 1 is slow, the only strategy for achieving very low latency is state channels, which often have high capital requirements and can be difficult to generalize. State channels will always beat layer 1 blockchains in latency as state channels require only a single network message, but in those cases where state channels do not work well, layer 1 blockchains can still come closer than they do today.

Hence, the other extreme position, that blockchain base layers can be truly absolutely minimal, and not bother with either a quasi-Turing-complete execution engine or scalability to beyond the capacity of a single node, is also clearly false; there is a certain minimal level of complexity that is required for base layers to be powerful enough for applications to build on top of them, and we have not yet reached that level. Additional complexity is needed, though it should be chosen very carefully to make sure that it is maximally general purpose, and not targeted toward specific applications or technologies that will go out of fashion in two years due to loss of interest or better alternatives.

And even in the future base layers will need to continue to make some upgrades, especially if new technologies (eg. STARKs reaching higher levels of maturity) allow them to achieve stronger properties than they could before, though developers today can take care to make base layer platforms maximally forward-compatible with such potential improvements. So it will continue to be true that a balance between layer 1 and layer 2 improvements is needed to continue improving scalability, privacy and versatility, though layer 2 will continue to take up a larger and larger share of the innovation over time.

Update 2018.08.29: Justin Drake pointed out to me another good reason why some features may be best implemented on layer 1: those features are public goods, and so could not be efficiently or reliably funded with feature-specific use fees, and hence are best paid for by subsidies paid out of issuance or burned transaction fees. One possible example of this is secure random number generation, and another is generation of zero knowledge proofs for more efficient client validation of correctness of various claims about blockchain contents or state.

A Guide to 99% Fault Tolerant Consensus

2018 Aug 07

Special thanks to Emin Gun Sirer for review

We've heard for a long time that it's possible to achieve consensus with 50% fault tolerance in a synchronous network where messages broadcasted by any honest node are guaranteed to be received by all other honest nodes within some known time period (if an attacker has *more* than 50%, they can perform a "51% attack", and there's an analogue of this for any algorithm of this type). We've also heard for a long time that if you want to relax the synchrony assumption, and have an algorithm that's "safe under asynchrony", the maximum achievable fault tolerance drops to 33% (PBFT, Casper FFG, etc all fall into this category). But did you know that if you add *even more* assumptions (specifically, you require *observers*, ie. users that are not actively participating in the consensus but care about its output, to also be actively watching the consensus, and not just downloading its output after the fact), you can increase fault tolerance all the way to 99%?

This has in fact been known for a long time; Leslie Lamport's famous 1982 paper "The Byzantine Generals Problem" (link here) contains a description of the algorithm. The following will be my attempt to describe and reformulate the algorithm in a simplified form.

Suppose that there are N consensus-participating nodes, and everyone agrees who these nodes are ahead of time (depending on context, they could have been selected by a trusted party or, if stronger decentralization is desired, by some proof of work or proof of stake scheme). We label these nodes $0 ... N-1$. Suppose also that there is a known bound D on network latency plus clock disparity (eg. D = 8 seconds). Each node has the ability to publish a value at time T (a malicious node can of course propose values earlier or later than T). All nodes wait $(N-1) \cdot D$ seconds, running the following process. Define $x : i$ as "the value x signed by node i", $x : i : j$ as "the value x signed by i, and that value and signature together signed by j", etc. The proposals published in the first stage will be of the form $v: i$ for some v and i, containing the signature of the node that proposed it.

If a validator i receives some message $v : i[1] : ... : i[k]$, where $i[1] ... i[k]$ is a list of indices that have (sequentially) signed the message already (just v by itself would count as $k=0$, and $(v:i)$ as $k=1$), then the validator checks that (i) the time is less than $T + k \cdot D$, and (ii) they have not yet seen a valid message containing v; if both checks pass, they publish $v : i[1] : ... : i[k] : i$.

At time $T + (N-1) \cdot D$, nodes stop listening. At this point, there is a guarantee that honest nodes have all "validly seen" the same set of values.

Node 1 (red) is malicious, and nodes 0 and 2 (grey) are honest. At the start, the two honest nodes make their proposals y and x, and the attacker proposes both w and z late. w reaches node 0 on time but not node 2, and z reaches neither node on time. At time $T + D$, nodes 0 and 2 rebroadcast all values they've seen that they have not yet broadcasted, but add their signatures on (x and w for node 0, y for node 2). Both honest nodes saw $\{x, y, w\}$.

If the problem demands choosing one value, they can use some "choice" function to pick a single value out of the values they have seen (eg. they take the one with the lowest hash). The nodes can then agree on this value.

Now, let's explore why this works. What we need to prove is that if one honest node has seen a particular value (validly), then every other honest node has also seen that value (and if we prove this, then we know that all honest nodes have seen the same set of values, and so if all honest nodes are running the same choice function, they will choose the same value). Suppose that any honest node receives a message $v : i[1] : ... : i[k]$ that they perceive to be valid (ie. it arrives before time $T + k \cdot D$). Suppose x is the index of a single other honest node. Either x is part of $\{i[1] ... i[k]\}$ or it is not.

- In the first case (say $x = i[j]$ for this message), we know that the honest node x had already broadcasted that message, and they did so in response to a message with $j-1$ signatures that they received before time $T + (j-1) \cdot D$, so they broadcast their message at that time, and so the message must have been received by all honest nodes before time $T + j \cdot D$.
- In the second case, since the honest node sees the message before time $T + k \cdot D$, then they will broadcast the message with their signature and guarantee that everyone, including x, will see it before time $T + (k+1) \cdot D$.

Notice that the algorithm uses the act of adding one's own signature as a kind of "bump" on the timeout of a message, and it's this ability that guarantees that if one honest node saw a message on time, they can ensure that everyone else sees the message on time as well, as the definition of "on time" increments by more than network latency with every added signature.

In the case where one node is honest, can we guarantee that passive *observers* (ie. non-consensus-participating nodes that care about knowing the outcome) can also see the outcome, even if we require them to be watching the process the whole time? With the scheme as written, there's a problem. Suppose that a commander and some subset of k (malicious) validators produce a message $v : i[1] : .... : i[k]$, and broadcast it directly to some "victims" just before time $T + k \cdot D$. The victims see the message as being "on time", but when they rebroadcast it, it only reaches all honest consensus-participating nodes after $T + k \cdot D$, and so all honest consensus-participating nodes reject it.

But we can plug this hole. We require D to be a bound on *two times* network latency plus clock disparity. We then put a different timeout on observers: an observer accepts $v : i[1] : .... : i[k]$ before time $T + (k - 0.5) \cdot D$. Now, suppose an observer sees a message an accepts it. They will be able to broadcast it to an honest node before time $T + k \cdot D$, and the honest node will issue the message with their signature attached, which will reach all other observers before time $T + (k + 0.5) \cdot D$, the timeout for messages with $k+1$ signatures.

Retrofitting onto other consensus algorithms

The above could theoretically be used as a standalone consensus algorithm, and could even be used to run a proof-of-stake blockchain. The validator set of round $N+1$ of the consensus could itself be decided during round N of the consensus (eg. each round of a consensus could also accept "deposit" and "withdraw" transactions, which if accepted and correctly signed would add or remove validators into the next round). The main additional ingredient that would need to be added is a mechanism for deciding who is allowed to propose blocks (eg. each round could have one designated proposer). It could also be modified to be usable as a proof-of-work blockchain, by allowing consensus-participating nodes to "declare themselves" in real time by publishing a proof of work solution on top of their public key at th same time as signing a message with it.

However, the synchrony assumption is very strong, and so we would like to be able to work without it in the case where we don't need more than 33% or 50% fault tolerance. There is a way to accomplish this. Suppose that we have some other consensus algorithm (eg. PBFT, Casper FFG, chain-based PoS) whose output *can* be seen by occasionally-online observers (we'll call this the *threshold-dependent* consensus algorithm, as opposed to the algorithm above, which we'll call the *latency-dependent* consensus algorithm). Suppose that the threshold-dependent consensus algorithm runs continuously, in a mode where it is constantly "finalizing" new blocks onto a chain (ie. each finalized value points to some previous finalized value as a "parent"; if there's a sequence of pointers $A \rightarrow ... \rightarrow B$, we'll call A a *descendant* of B).

We can retrofit the latency-dependent algorithm onto this structure, giving always-online observers access to a kind of "strong finality" on checkpoints, with fault tolerance ~95% (you can push this arbitrarily close to 100% by adding more validators and requiring the process to take longer).

Every time the time reaches some multiple of 4096 seconds, we run the latency-dependent algorithm, choosing 512 random nodes to participate in the algorithm. A valid proposal is any valid chain of values that were finalized by the threshold-dependent algorithm. If a node sees some finalized value before time $T + k \cdot D$ (D = 8 seconds) with k signatures, it accepts the chain into its set of known chains and rebroadcasts it with its own signature added; observers use a threshold of $T +

$(k - 0.5) \cdot D$) as before.

The "choice" function used at the end is simple:

- Finalized values that are not descendants of what was already agreed to be a finalized value in the previous round are ignored
- Finalized values that are invalid are ignored
- To choose between two valid finalized values, pick the one with the lower hash

If 5% of validators are honest, there is only a roughly 1 in 1 trillion chance that none of the 512 randomly selected nodes will be honest, and so as long as the network latency plus clock disparity is less than $\frac{D}{2}$ the above algorithm will work, correctly coordinating nodes on some single finalized value, even if multiple conflicting finalized values are presented because the fault tolerance of the threshold-dependent algorithm is broken.

If the fault tolerance of the threshold-dependent consensus algorithm is met (usually 50% or 67% honest), then the threshold-dependent consensus algorithm will either not finalize any new checkpoints, or it will finalize new checkpoints that are compatible with each other (eg. a series of checkpoints where each points to the previous as a parent), so even if network latency exceeds $\frac{D}{2}$ (or even D), and as a result nodes participating in the latency-dependent algorithm disagree on which value they accept, the values they accept are still guaranteed to be part of the same chain and so there is no actual disagreement. Once latency recovers back to normal in some future round, the latency-dependent consensus will get back "in sync".

If the assumptions of both the threshold-dependent and latency-dependent consensus algorithms are broken *at the same time* (or in consecutive rounds), then the algorithm can break down. For example, suppose in one round, the threshold-dependent consensus finalizes $Z \rightarrow Y \rightarrow X$ and the latency-dependent consensus disagrees between Y and X, and in the next round the threshold-dependent consensus finalizes a descendant W of X which is *not* a descendant of Y; in the latency-dependent consensus, the nodes who agreed Y will not accept W, but the nodes that agreed X will. However, this is unavoidable; the impossibility of safe-under-asynchrony consensus with more than $\frac{1}{3}$ fault tolerance is a [well known result](#) in Byzantine fault tolerance theory, as is the impossibility of more than $\frac{1}{2}$ fault tolerance even allowing synchrony assumptions but assuming offline observers.

STARKs, Part 3: Into the Weeds

2018 Jul 21

Special thanks to Eli ben Sasson for his kind assistance, as usual. Special thanks to Chih-Cheng Liang and Justin Drake for review, and to Ben Fisch for suggesting the reverse MIMC technique for a VDF (paper [here](#))

Trigger warning: math and lots of python

As a followup to [Part 1](#) and [Part 2](#) of this series, this post will cover what it looks like to actually implement a STARK, complete with an implementation in python. STARKs ("Scalable Transparent ARgument of Knowledge" are a technique for creating a proof that $f(x)=y$ where f may potentially take a very long time to calculate, but where the proof can be verified very quickly. A STARK is "doubly scalable": for a computation with t steps, it takes roughly $O(t \cdot \log{t})$ steps to produce a proof, which is likely optimal, and it takes ~$O(\log^2{t})$ steps to verify, which for even moderately large values of t is much faster than the original computation. STARKs can also have a privacy-preserving "zero knowledge" property, though the use case we will apply them to here, making verifiable delay functions, does not require this property, so we do not need to worry about it.

First, some disclaimers:

- This code has not been thoroughly audited; soundness in production use cases is not guaranteed
- This code is very suboptimal (it's written in Python, what did you expect)
- STARKs "in real life" (ie. as implemented in Eli and co's production implementations) tend to use binary fields and not prime fields for application-specific efficiency reasons; however, they do stress in their writings the prime field-based approach to STARKs described here is legitimate and can be used
- There is no "one true way" to do a STARK. It's a broad category of cryptographic and mathematical constructs, with different setups optimal for different applications and constant ongoing research to reduce prover and verifier complexity and improve soundness.
- This article absolutely expects you to know how modular arithmetic and prime fields work, and be comfortable with the concepts of polynomials, interpolation and evaluation. If you don't, go back to [Part 2](#), and also this [earlier post on quadratic arithmetic programs](#)

Now, let's get to it.

MIMC

Here is the function we'll be doing a STARK of:

```
def mimc(inp, steps, round_constants):
    start_time = time.time()
    for i in range(steps-1):
        inp = (inp**3 + round_constants[i % len(round_constants)]) % modulus
    print("MIMC computed in %.4f sec" % (time.time() - start_time))
    return inp
```

We choose MIMC (see [paper](#)) as the example because it is both (i) simple to understand and (ii) interesting enough to be useful in real life. The function can be viewed visually as follows:
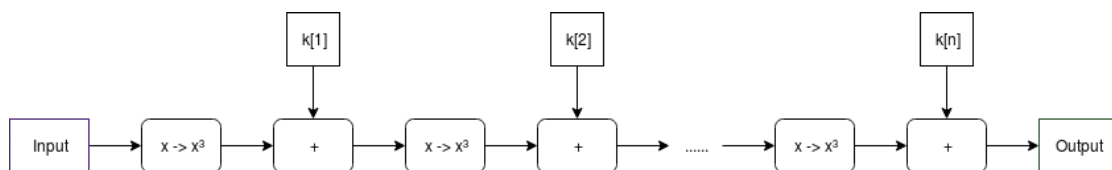
Note: in many discussions of MIMC, you will typically see XOR used instead of +; this is because MIMC is typically done over binary fields, where addition is XOR; here we are doing it over prime fields.

In our example, the round constants will be a relatively small list (eg. 64 items) that gets cycled through over and over again (that is, after `k[64]` it loops back to using `k[1]`).

MIMC with a very large number of rounds, as we're doing here, is useful as a *verifiable delay function* - a function which is difficult to compute, and particularly non-parallelizable to compute, but relatively easy to verify. MIMC by itself achieves this property to some extent because MIMC *can* be computed "backward" (recovering the "input" from its corresponding "output"), but computing it backward takes about 100 times longer to compute than the forward direction (and neither direction can be significantly sped up by parallelization). So you can think of computing the function in the backward direction as being the act of "computing" the non-parallelizable proof of work, and computing the function in the forward direction as being the process of "verifying" it.
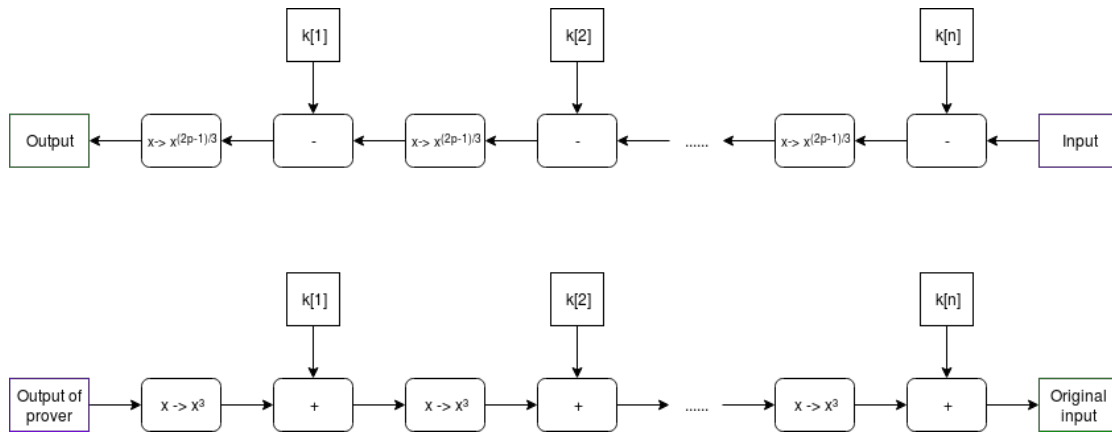
\(x \rightarrow x^{(2p-1)/3}\) gives the inverse of \(x \rightarrow x^3\); this is true because of [Fermat's Little Theorem](#), a theorem that despite its supposed littleness is arguably much more important to mathematics than Fermat's more famous "Last Theorem".

What we will try to achieve here is to make verification much more efficient by using a STARK - instead of the verifier having to run MIMC in the forward direction themselves, the prover, after completing the computation in the "backward direction", would compute a STARK of the computation in the "forward direction", and the verifier would simply verify the STARK. The hope is that the overhead of computing a STARK can be less than the difference in speed running MIMC forwards relative to backwards, so a prover's time would still be dominated by the initial "backward" computation, and not the (highly parallelizable) STARK computation. Verification of a STARK can be relatively fast (in our python implementation, ~0.05-0.3 seconds), no matter how long the original computation is.

All calculations are done modulo \(2^{256} - 351 \cdot 2^{32} + 1\); we are using this prime field modulus because it is the largest prime below \(2^{256}\) whose multiplicative group contains an order \(2^{32}\) subgroup (that is, there's a number \(g\) such that successive powers of \(g\) modulo this prime loop around back to \(1\) after exactly \(2^{32}\) cycles), and which is of the form \(6k+5\). The first property is necessary to make sure that our efficient versions of the FFT and FRI algorithms can work, and the second ensures that MIMC actually can be computed "backwards" (see the use of \(x \rightarrow x^{(2p-1)/3}\) above).

Prime field operations

We start off by building a convenience class that does prime field operations, as well as operations with polynomials over prime fields. The code is [here](#). First some trivial bits:

```
class PrimeField():
    def __init__(self, modulus):
        # Quick primality test
        assert pow(2, modulus, modulus) == 2
        self.modulus = modulus

    def add(self, x, y):
        return (x+y) % self.modulus

    def sub(self, x, y):
        return (x-y) % self.modulus

    def mul(self, x, y):
        return (x*y) % self.modulus
```

And the [Extended Euclidean Algorithm](#) for computing modular inverses (the equivalent of computing \(\frac{1}{x}\)) in a prime field):

```
# Modular inverse using the extended Euclidean algorithm
def inv(self, a):
    if a == 0:
        return 0
    lm, hm = 1, 0
    low, high = a % self.modulus, self.modulus
    while low > 1:
        r = high//low
        nm, new = hm-lm*r, high-low*r
        lm, low, hm, high = nm, new, lm, low
    return lm % self.modulus
```

The above algorithm is relatively expensive; fortunately, for the special case where we need to do many modular inverses, there's a simple mathematical trick that allows us to compute many inverses, called [Montgomery batch inversion](#):
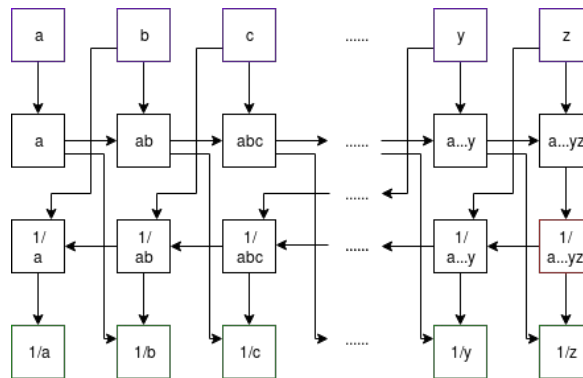
*Using Montgomery batch inversion to compute modular inverses. Inputs purple, outputs green, multiplication gates black; the red square is the **only** modular inversion.*

The code below implements this algorithm, with some slightly ugly special case logic so that if there are zeroes in the set of what we are inverting, it sets their inverse to 0 and moves along.

```
def multi_inv(self, values):
    partials = [1]
    for i in range(len(values)):
        partials.append(self.mul(partials[-1], values[i] or 1))
    inv = self.inv(partials[-1])
    outputs = [0] * len(values)
    for i in range(len(values), 0, -1):
        outputs[i-1] = self.mul(partials[i-1], inv) if values[i-1] else 0
        inv = self.mul(inv, values[i-1] or 1)
    return outputs
```

This batch inverse algorithm will prove important later on, when we start dealing with dividing sets of evaluations of polynomials.

Now we move on to some polynomial operations. We treat a polynomial as an array, where element i is the ith degree term (eg. $x^{3} + 2x + 1$ becomes [1, 2, 0, 1]). Here's the operation of evaluating a polynomial at *one point*:

```
# Evaluate a polynomial at a point
def eval_poly_at(self, p, x):
    y = 0
    power_of_x = 1
    for i, p_coeff in enumerate(p):
        y += power_of_x * p_coeff
        power_of_x = (power_of_x * x) % self.modulus
    return y % self.modulus
```

Challenge
What is the output of f.eval_poly_at([4, 5, 6], 2) if the modulus is 31?

Mouseover below for answer

There is also code for adding, subtracting, multiplying and dividing polynomials; this is textbook long addition/subtraction/multiplication/division. The one non-trivial thing is Lagrange interpolation, which takes as input a set of x and y coordinates, and returns the minimal polynomial that passes through all of those points (you can think of it as being the inverse of polynomial evaluation):

```
# Build a polynomial that returns 0 at all specified xs
def zpoly(self, xs):
    root = [1]
    for x in xs:
        root.insert(0, 0)
        for j in range(len(root)-1):
            root[j] -= root[j+1] * x
    return [x % self.modulus for x in root]

def lagrange_interp(self, xs, ys):
    # Generate master numerator polynomial, eg. (x - x1) * (x - x2) * ... * (x - xn)
    root = self.zpoly(xs)

    # Generate per-value numerator polynomials, eg. for x=x2,
    # (x - x1) * (x - x3) * ... * (x - xn), by dividing the master
    # polynomial back by each x coordinate
    nums = [self.div_polys(root, [-x, 1]) for x in xs]
```

```
        # Generate denominators by evaluating numerator polys at each x
        denoms = [self.eval_poly_at(nums[i], xs[i]) for i in range(len(xs))]
        invdenoms = self.multi_inv(denoms)

        # Generate output polynomial, which is the sum of the per-value numerator
        # polynomials rescaled to have the right y values
        b = [0 for y in ys]
        for i in range(len(xs)):
            yslice = self.mul(ys[i], invdenoms[i])
            for j in range(len(ys)):
                if nums[i][j] and ys[i]:
                    b[j] += nums[i][j] * yslice
        return [x % self.modulus for x in b]
```

See the "M of N" section of this article for a description of the math. Note that we also have special-case methods `lagrange_interp_4` and `lagrange_interp_2` to speed up the very frequent operations of Lagrange interpolation of degree < 2 and degree < 4 polynomials.

Fast Fourier Transforms

If you read the above algorithms carefully, you might notice that Lagrange interpolation and multi-point evaluation (that is, evaluating a degree $< N$ polynomial at N points) both take quadratic time to execute, so for example doing a Lagrange interpolation of one thousand points takes a few million steps to execute, and a Lagrange interpolation of one million points takes a few trillion. This is an unacceptably high level of inefficiency, so we will use a more efficient algorithm, the Fast Fourier Transform.

The FFT only takes $O(n \cdot log(n))$ time (ie. ~10,000 steps for 1,000 points, ~20 million steps for 1 million points), though it is more restricted in scope; the x coordinates must be a complete set of **roots of unity** of some **order** $N = 2^{k}$. That is, if there are N points, the x coordinates must be successive powers $1, p, p^{2}, p^{3}$... of some p where $p^{N} = 1$. The algorithm can, surprisingly enough, be used for multi-point evaluation *or* interpolation, with one small parameter tweak.

   Challenge Find a 16th root of unity mod 337 that is not an 8th root of unity.

   Mouseover below for answer

Here's the algorithm (in a slightly simplified form; see code here for something slightly more optimized):

```
def fft(vals, modulus, root_of_unity):
    if len(vals) == 1:
        return vals
    L = fft(vals[::2], modulus, pow(root_of_unity, 2, modulus))
    R = fft(vals[1::2], modulus, pow(root_of_unity, 2, modulus))
    o = [0 for i in vals]
    for i, (x, y) in enumerate(zip(L, R)):
        y_times_root = y*pow(root_of_unity, i, modulus)
        o[i] = (x+y_times_root) % modulus
        o[i+len(L)] = (x-y_times_root) % modulus
    return o

def inv_fft(vals, modulus, root_of_unity):
    f = PrimeField(modulus)
    # Inverse FFT
    invlen = f.inv(len(vals))
    return [(x*invlen) % modulus for x in
            fft(vals, modulus, f.inv(root_of_unity))]
```

You can try running it on a few inputs yourself and check that it gives results that, when you use `eval_poly_at` on them, give you the answers you expect to get. For example:

```
>>> fft.fft([3,1,4,1,5,9,2,6], 337, 85, inv=True)
[46, 169, 29, 149, 126, 262, 140, 93]
>>> f = poly_utils.PrimeField(337)
>>> [f.eval_poly_at([46, 169, 29, 149, 126, 262, 140, 93], f.exp(85, i)) for i in range(8)]
[3, 1, 4, 1, 5, 9, 2, 6]
```

A Fourier transform takes as input [x[0] .... x[n-1]], and its goal is to output x[0] + x[1] + ... + x[n-1] as the first element, x[0] + x[1] * 2 + ... + x[n-1] * w**(n-1) as the second element, etc etc; a fast Fourier transform accomplishes this by splitting the data in half, doing an FFT on both halves, and then gluing the result back together.
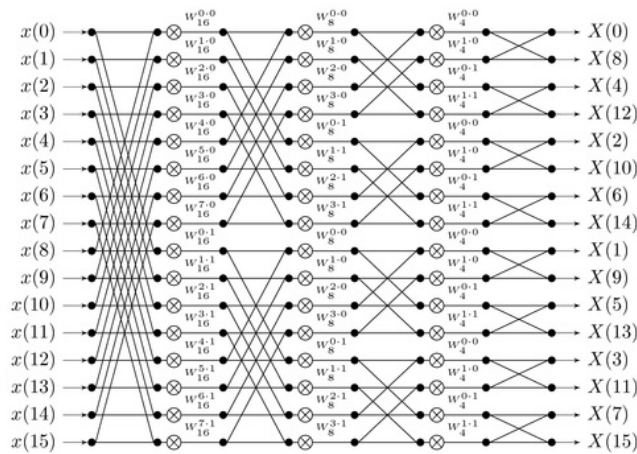
A diagram of how information flows through the FFT computation. Notice how the FFT consists of a "gluing" step followed by two copies of the FFT on two halves of the data, and so on recursively until you're down to one element.

I recommend [this](#) for more intuition on how or why the FFT works and polynomial math in general, and [this thread](#) for some more specifics on DFT vs FFT, though be warned that most literature on Fourier transforms talks about Fourier transforms over *real and complex numbers*, not *prime fields*. If you find this too hard and don't want to understand it, just treat it as weird spooky voodoo that just works because you ran the code a few times and verified that it works, and you'll be fine too.

Thank Goodness It's FRI-day (that's "Fast Reed-Solomon Interactive Oracle Proofs of Proximity")

Reminder: *now may be a good time to review and re-read [Part 2](#)*

Now, we'll get into [the code](#) for making a low-degree proof. To review, a low-degree proof is a (probabilistic) proof that at least some high percentage (eg. 80%) of a given set of values represent the evaluations of some specific polynomial whose degree is much lower than the number of values given. Intuitively, just think of it as a proof that "some Merkle root that we claim represents a polynomial actually does represent a polynomial, possibly with a few errors". As input, we have:

- A set of values that we claim are the evaluation of a low-degree polynomial
- A root of unity; the x coordinates at which the polynomial is evaluated are successive powers of this root of unity
- A value N such that we are proving the degree of the polynomial is *strictly less than* N
- The modulus

Our approach is a recursive one, with two cases. First, if the degree is low enough, we just provide the entire list of values as a proof; this is the "base case". Verification of the base case is trivial: do an FFT or Lagrange interpolation or whatever else to interpolate the polynomial representing those values, and verify that its degree is $< N$. Otherwise, if the degree is higher than some set minimum, we do the vertical-and-diagonal trick described [at the bottom of Part 2](#).

We start off by putting the values into a Merkle tree and using the Merkle root to select a pseudo-random x coordinate (special_x). We then calculate the "column":

```
# Calculate the set of x coordinates
xs = get_power_cycle(root_of_unity, modulus)

column = []
for i in range(len(xs)//4):
    x_poly = f.lagrange_interp_4(
        [xs[i+len(xs)*j//4] for j in range(4)],
        [values[i+len(values)*j//4] for j in range(4)],
    )
    column.append(f.eval_poly_at(x_poly, special_x))
```

This packs a lot into a few lines of code. The broad idea is to re-interpret the polynomial $P(x)$ as a polynomial $Q(x, y)$, where $P(x) = Q(x, x^4)$. If P has degree $< N$, then $P'(y) = Q(special\_x, y)$ will have degree $< \frac{N}{4}$. Since we don't want to take the effort to actually compute Q in coefficient form (that would take a still-relatively-nasty-and-expensive FFT!), we instead use another trick. For any given value of x^4, there are 4 corresponding values of x: x, $modulus - x$, and x multiplied by the two modular square roots of -1. So we already have four values of $Q(?, x^4)$, which we can use to interpolate the polynomial $R(x) = Q(x, x^4)$, and from there calculate $R(special\_x) = Q(special\_x, x^4) = P'(x^4)$. There are $\frac{N}{4}$ possible values of x^4, and this lets us easily calculate all of them.
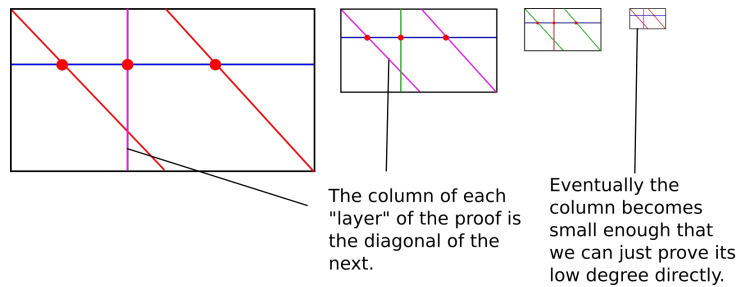
A diagram from part 2; it helps to keep this in mind when understanding what's going on here

Our proof consists of some number (eg. 40) of random queries from the list of values of x^4 (using the Merkle root of the column as a seed), and for each query we provide Merkle branches of the five values of $Q(?, x^4)$:

```
m2 = merkelize(column)

# Pseudo-randomly select y indices to sample
# (m2[1] is the Merkle root of the column)
ys = get_pseudorandom_indices(m2[1], len(column), 40)

# Compute the Merkle branches for the values in the polynomial and the column
branches = []
for y in ys:
    branches.append([mk_branch(m2, y)] +
                    [mk_branch(m, y + (len(xs) // 4) * j) for j in range(4)])
```

The verifier's job will be to verify that these five values actually do lie on the same degree < 4 polynomial. From there, we recurse and do an FRI on the column, verifying that the column actually does have degree $< \frac{N}{4}$. That really is all there is to FRI.

As a challenge exercise, you could try creating low-degree proofs of polynomial evaluations that have errors in them, and see how many errors you can get away passing the verifier with (hint, you'll need to modify the `prove_low_degree` function; with the default prover, even one error will balloon up and cause verification to fail).

The STARK

Reminder: *now may be a good time to review and re-read [Part 1](#)*

Now, we get to the actual meat that puts all of these pieces together: `def mk_mimc_proof(inp, steps, round_constants)` (code [here](#)), which generates a proof of the execution result of running the MIMC function with the given input for some number of steps. First, some asserts:

```
assert steps <= 2**32 // extension_factor
assert is_a_power_of_2(steps) and is_a_power_of_2(len(round_constants))
assert len(round_constants) < steps
```

The extension factor is the extent to which we will be "stretching" the computational trace (the set of "intermediate values" of executing the MIMC function). We need the step count multiplied by the extension factor to be at most 2^{32}, because we don't have roots of unity of order 2^k for $k > 32$.

Our first computation will be to generate the computational trace; that is, all of the *intermediate* values of the computation, from the input going all the way to the output.

```
# Generate the computational trace
computational_trace = [inp]
for i in range(steps-1):
    computational_trace.append((computational_trace[-1]**3 + round_constants[i % len(round_constants)]) % modulus)
output = computational_trace[-1]
```

We then convert the computation trace into a polynomial, "laying down" successive values in the trace on successive powers of a root of unity g where $g^{steps} = 1$, and we then evaluate the polynomial in a larger set, of successive powers of a root of unity g_2 where $(g_2)^{steps \cdot 8} = 1$ (note that $(g_2)^8 = g$).

```
computational_trace_polynomial = inv_fft(computational_trace, modulus, subroot)
p_evaluations = fft(computational_trace_polynomial, modulus, root_of_unity)
```
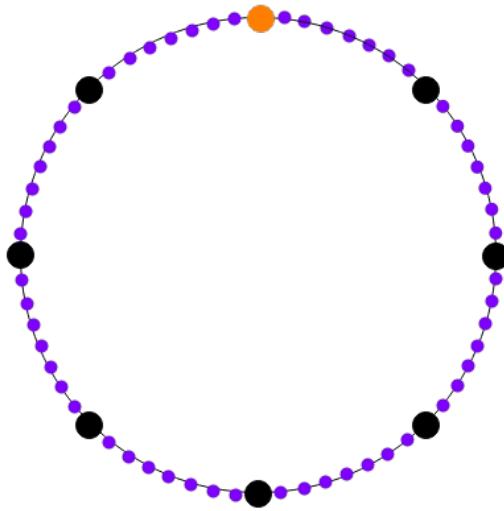
Black: powers of (g_1). Purple: powers of (g_2). Orange: 1. You can look at successive roots of unity as being arranged in a circle in this way. We are "laying" the computational trace along powers of (g_1), and then extending it compute the values of the same polynomial at the intermediate values (ie. the powers of (g_2)).

We can convert the round constants of MIMC into a polynomial. Because these round constants loop around very frequently (in our tests, every 64 steps), it turns out that they form a degree-64 polynomial, and we can fairly easily compute its expression, and its extension:

```
skips2 = steps // len(round_constants)
constants_mini_polynomial = fft(round_constants, modulus, f.exp(subroot, skips2), inv=True)
constants_polynomial = [0 if i % skips2 else constants_mini_polynomial[i//skips2] for i in range(steps)]
constants_mini_extension = fft(constants_mini_polynomial, modulus, f.exp(root_of_unity, skips2))
```

Suppose there are 8192 steps of execution and 64 round constants. Here is what we are doing: we are doing an FFT to compute the round constants *as a function of $((g\_1)^{128})$*. We then add zeroes in between the constants to make it a function of (g_1) itself. Because $((g\_1)^{128})$ loops around every 64 steps, we know this function of (g_1) will as well. We only compute 512 steps of the extension, because we know that the extension repeats after 512 steps as well.

We now, as in the Fibonacci example in Part 1, calculate $(C(P(x)))$, except this time it's $(C(P(x), P(g\_1 \cdot x), K(x)))$:

```
# Create the composed polynomial such that
# C(P(x), P(g1*x), K(x)) = P(g1*x) - P(x)**3 - K(x)
c_of_p_evaluations = [(p_evaluations[(i+extension_factor)%precision] -
                        f.exp(p_evaluations[i], 3) -
                        constants_mini_extension[i % len(constants_mini_extension)])
                    % modulus for i in range(precision)]
print('Computed C(P, K) polynomial')
```

Note that here we are no longer working with polynomials in *coefficient form*; we are working with the polynomials in terms of their evaluations at successive powers of the higher-order root of unity.

c_of_p is intended to be $(Q(x) = C(P(x), P(g\_1 \cdot x), K(x)) = P(g\_1 \cdot x) - P(x)^3 - K(x))$; the goal is that for every (x) that we are laying the computational trace along (except for the last step, as there's no step "after" the last step), the next value in the trace is equal to the previous value in the trace cubed, plus the round constant. Unlike the Fibonacci example in Part 1, where if one computational step was at coordinate (k), the next step is at coordinate $(k+1)$, here we are laying down the computational trace along successive powers of the lower-order root of unity (g_1), so if one computational step is located at $(x = (g\_1)^i)$, the "next" step is located at $((g\_1)^{i+1}) = ((g\_1)^i \cdot g\_1 = x \cdot g\_1)$. Hence, for every power of the lower-order root of unity (g_1) (except the last), we want it to be the case that $(P(x \cdot g\_1) = P(x)^3 + K(x))$, or $(P(x \cdot g\_1) - P(x)^3 - K(x) = Q(x) = 0)$. Thus, $(Q(x))$ will be equal to zero at all successive powers of the lower-order root of unity (g) (except the last).

There is an algebraic theorem that proves that if $(Q(x))$ is equal to zero at all of these x coordinates, then it is a multiple of the *minimal* polynomial that is equal to zero at all of these x coordinates: $(Z(x) = (x - x\_1) \cdot (x - x\_2) \cdot ... \cdot (x - x\_n))$. Since proving that $(Q(x))$ is equal to zero at every single coordinate we want to check is too hard (as verifying such a proof would take longer than just running the original computation!), instead we use an indirect approach to (probabilistically) prove that $(Q(x))$ is a multiple of $(Z(x))$. And how do we do that? By providing the quotient $(D(x) = \frac{Q(x)}{Z(x)})$ and using FRI to prove that it's an actual polynomial and not a fraction, of course!

We chose the particular arrangement of lower and higher order roots of unity (rather than, say, laying the computational trace along the first few powers of the higher order root of unity) because it turns out that computing $

$(Z(x)\)$ (the polynomial that evaluates to zero at all points along the computational trace except the last), and dividing by $\(Z(x)\)$ is trivial there: the expression of $\(Z\)$ is a fraction of two terms.

```
# Compute D(x) = Q(x) / Z(x)
# Z(x) = (x^steps - 1) / (x - x_atlast_step)
z_num_evaluations = [xs[(i * steps) % precision] - 1 for i in range(precision)]
z_num_inv = f.multi_inv(z_num_evaluations)
z_den_evaluations = [xs[i] - last_step_position for i in range(precision)]
d_evaluations = [cp * zd * zni % modulus for cp, zd, zni in zip(c_of_p_evaluations, z_den_evaluations, z_num_inv)]
print('Computed D polynomial')
```

Notice that we compute the numerator and denominator of $\(Z\)$ directly in "evaluation form", and then use the batch modular inversion to turn dividing by $\(Z\)$ into a multiplication ($\(\cdot z\_d \cdot z\_ni\)$), and then pointwise multiply the evaluations of $\(Q(x)\)$ by these inverses of $\(Z(x)\)$. Note that at the powers of the lower-order root of unity except the last (ie. along the portion of the low-degree extension that is part of the original computational trace), $\(Z(x) = 0\)$, so this computation involving its inverse will break. This is unfortunate, though we will plug the hole by simply modifying the random checks and FRI algorithm to not sample at those points, so the fact that we calculated them wrong will never matter.

Because $\(Z(x)\)$ can be expressed so compactly, we get another benefit: the verifier can compute $\(Z(x)\)$ for any specific $\(x\)$ extremely quickly, without needing any precomputation. It's okay for the *prover* to have to deal with polynomials whose size equals the number of steps, but we don't want to ask the *verifier* to do the same, as we want verification to be succinct (ie. ultra-fast, with proofs as small as possible).

Probabilistically checking $\(D(x) \cdot Z(x) = Q(x)\)$ at a few randomly selected points allows us to verify the **transition constraints** - that each computational step is a valid consequence of the previous step. But we also want to verify the **boundary constraints** - that the input and the output of the computation is what the prover says they are. Just asking the prover to provide evaluations of $\(P(1)\)$, $\(D(1)\)$, $\(P(last\_step)\)$ and $\(D(last\_step)\)$ (where $\(last\_step\)$ (or $\(g^{steps-1}\)$) is the coordinate corresponding to the last step in the computation) is too fragile; there's no proof that those values are on the same polynomial as the rest of the data. So instead we use a similar kind of polynomial division trick:

```
# Compute interpolant of ((1, input), (x_atlast_step, output))
interpolant = f.lagrange_interp_2([1, last_step_position], [inp, output])
i_evaluations = [f.eval_poly_at(interpolant, x) for x in xs]

zeropoly2 = f.mul_polys([-1, 1], [-last_step_position, 1])
inv_z2_evaluations = f.multi_inv([f.eval_poly_at(quotient, x) for x in xs])

# B = (P - I) / Z2
b_evaluations = [((p - i) * invq) % modulus for p, i, invq in zip(p_evaluations, i_evaluations, inv_z2_evaluations)]
print('Computed B polynomial')
```

The argument is as follows. The prover wants to prove $\(P(1) = input\)$ and $\(P(last\_step) = output\)$. If we take $\(I(x)\)$ as the *interpolant* - the line that crosses the two points $\((1, input)\)$ and $\((last\_step, output)\)$, then $\(P(x) - I(x)\)$ would be equal to zero at those two points. Thus, it suffices to prove that $\(P(x) - I(x)\)$ is a multiple of $\((x - 1) \cdot (x - last\_step)\)$, and we do that by... providing the quotient!
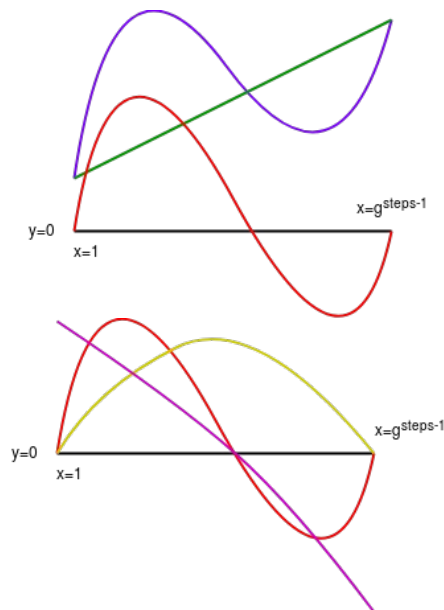
Purple: computational trace polynomial (P). Green: interpolant (I) (notice how the interpolant is constructed to equal the input (which should be the first step of the computational trace) at x=1 and the output (which should be the last step of the computational trace) at $\(x=g^{steps-1}\)$). Red: $\(P - I\)$. Yellow: the minimal polynomial that equals $\(0\)$ at $\(x=1\)$ and $\(x=g^{steps-1}\)$ (that is, $\(Z\_2\)$). Pink: $\(\frac{P - I}{Z\_2}\)$.

Challenge Suppose you wanted to *also* prove that the value in the computational trace after the 703rd computational step is equal to 8018284612598740. How would you modify the above algorithm to do that?
Mouseover below for answer

Now, we commit to the Merkle root of \(P\), \(D\) and \(B\) combined together.

```
# Compute their Merkle roots
mtree = merkelize([pval.to_bytes(32, 'big') +
                   dval.to_bytes(32, 'big') +
                   bval.to_bytes(32, 'big') for
                   pval, dval, bval in zip(p_evaluations, d_evaluations, b_evaluations)])
print('Computed hash root')
```

Now, we need to prove that \(P\), \(D\) and \(B\) are all actually polynomials, and of the right max-degree. But FRI proofs are big and expensive, and we don't want to have three FRI proofs. So instead, we compute a pseudorandom linear combination of \(P\), \(D\) and \(B\) (using the Merkle root of \(P\), \(D\) and \(B\) as a seed), and do an FRI proof on that:

```
k1 = int.from_bytes(blake(mtree[1] + b'\x01'), 'big')
k2 = int.from_bytes(blake(mtree[1] + b'\x02'), 'big')
k3 = int.from_bytes(blake(mtree[1] + b'\x03'), 'big')
k4 = int.from_bytes(blake(mtree[1] + b'\x04'), 'big')

# Compute the linear combination. We don't even bother calculating it
# in coefficient form; we just compute the evaluations
root_of_unity_to_the_steps = f.exp(root_of_unity, steps)
powers = [1]
for i in range(1, precision):
    powers.append(powers[-1] * root_of_unity_to_the_steps % modulus)

l_evaluations = [(d_evaluations[i] +
                  p_evaluations[i] * k1 + p_evaluations[i] * k2 * powers[i] +
                  b_evaluations[i] * k3 + b_evaluations[i] * powers[i] * k4) % modulus
                  for i in range(precision)]
```

Unless all three of the polynomials have the right low degree, it's almost impossible that a randomly selected linear combination of them will (you have to get *extremely* lucky for the terms to cancel), so this is sufficient.

We want to prove that the degree of D is less than \(2 \cdot steps\), and that of \(P\) and \(B\) are less than \(steps\), so we actually make a random linear combination of \(P\), \(P \cdot x^{steps}\), \(B\), \(B^{steps}\) and \(D\), and check that the degree of this combination is less than \(2 \cdot steps\).

Now, we do some spot checks of all of the polynomials. We generate some random indices, and provide the Merkle branches of the polynomial evaluated at those indices:

```
# Do some spot checks of the Merkle tree at pseudo-random coordinates, excluding
# multiples of `extension_factor`
branches = []
samples = spot_check_security_factor
positions = get_pseudorandom_indices(l_mtree[1], precision, samples,
                                     exclude_multiples_of=extension_factor)
for pos in positions:
    branches.append(mk_branch(mtree, pos))
    branches.append(mk_branch(mtree, (pos + skips) % precision))
    branches.append(mk_branch(l_mtree, pos))
print('Computed %d spot checks' % samples)
```

The `get_pseudorandom_indices` function returns some random indices in the range [0...precision-1], and the `exclude_multiples_of` parameter tells it to not give values that are multiples of the given parameter (here, `extension_factor`). This ensures that we do not sample along the original computational trace, where we are likely to get wrong answers.

The proof (~250-500 kilobytes altogether) consists of a set of Merkle roots, the spot-checked branches, and a low-degree proof of the random linear combination:

```
o = [mtree[1],
     l_mtree[1],
     branches,
     prove_low_degree(l_evaluations, root_of_unity, steps * 2, modulus, exclude_multiples_of=extension_factor)]
```

The largest parts of the proof in practice are the Merkle branches, and the FRI proof, which consists of even more branches. And here's the "meat" of the verifier:

```
for i, pos in enumerate(positions):
    x = f.exp(G2, pos)
    x_to_the_steps = f.exp(x, steps)
    mbranch1 = verify_branch(m_root, pos, branches[i*3])
    mbranch2 = verify_branch(m_root, (pos+skips)%precision, branches[i*3+1])
```

```
l_of_x = verify_branch(l_root, pos, branches[i*3 + 2], output_as_int=True)

p_of_x = int.from_bytes(mbranch1[:32], 'big')
p_of_g1x = int.from_bytes(mbranch2[:32], 'big')
d_of_x = int.from_bytes(mbranch1[32:64], 'big')
b_of_x = int.from_bytes(mbranch1[64:], 'big')

zvalue = f.div(f.exp(x, steps) - 1,
               x - last_step_position)
k_of_x = f.eval_poly_at(constants_mini_polynomial, f.exp(x, skips2))

# Check transition constraints Q(x) = Z(x) * D(x)
assert (p_of_g1x - p_of_x ** 3 - k_of_x - zvalue * d_of_x) % modulus == 0

# Check boundary constraints B(x) * Z2(x) + I(x) = P(x)
interpolant = f.lagrange_interp_2([1, last_step_position], [inp, output])
zeropoly2 = f.mul_polys([-1, 1], [-last_step_position, 1])
assert (p_of_x - b_of_x * f.eval_poly_at(zeropoly2, x) -
        f.eval_poly_at(interpolant, x)) % modulus == 0

# Check correctness of the linear combination
assert (l_of_x - d_of_x -
        k1 * p_of_x - k2 * p_of_x * x_to_the_steps -
        k3 * b_of_x - k4 * b_of_x * x_to_the_steps) % modulus == 0
```

At every one of the positions that the prover provides a Merkle proof for, the verifier checks the Merkle proof, and checks that $C(P(x), P(g_1 \cdot x), K(x)) = Z(x) \cdot D(x)$ and $B(x) \cdot Z_2(x) + I(x) = P(x)$ (reminder: for x that are not along the original computation trace, $Z(x)$ will not be zero, and so $C(P(x), P(g_1 \cdot x), K(x))$ likely will not evaluate to zero). The verifier also checks that the linear combination is correct, and calls `verify_low_degree_proof(l_root, root_of_unity, fri_proof, steps * 2, modulus, exclude_multiples_of=extension_factor)` to verify the FRI proof. **And we're done**!

Well, not really; soundness analysis to prove how many spot-checks for the cross-polynomial checking and for the FRI are necessary is really tricky. But that's all there is to the code, at least if you don't care about making even crazier optimizations. When I run the code above, we get a STARK proving "overhead" of about 300-400x (eg. a MIMC computation that takes 0.2 seconds to calculate takes 60 second to prove), suggesting that with a 4-core machine computing the STARK of the MIMC computation in the forward direction could actually be faster than computing MIMC in the backward direction. That said, these are both relatively inefficient implementations in python, and the proving to running time ratio for properly optimized implementations may be different. Also, it's worth pointing out that the STARK proving overhead for MIMC is remarkably low, because MIMC is almost perfectly "arithmetizable" - it's mathematical form is very simple. For "average" computations, which contain less arithmetically clean operations (eg. checking if a number is greater or less than another number), the overhead is likely much higher, possibly around 10000-50000x.

On Radical Markets

2018 Apr 20

Recently I had the fortune to have received an advance copy of Eric Posner and Glen Weyl's new book, _Radical Markets_, which could be best described as an interesting new way of looking at the subject that is sometimes called "political economy" - tackling the big questions of how markets and politics and society intersect. The general philosophy of the book, as I interpret it, can be expressed as follows. Markets are great, and price mechanisms are an awesome way of guiding the use of resources in society and bringing together many participants' objectives and information into a coherent whole. However, markets are socially constructed because they depend on property rights that are socially constructed, and there are many different ways that markets and property rights can be constructed, some of which are unexplored and potentially far better than what we have today. Contra doctrinaire libertarians, freedom is a high-dimensional design space.

The book interests me for multiple reasons. First, although I spend most of my time in the blockchain/crypto space heading up the Ethereum project and in some cases providing various kinds of support to projects in the space, I do also have broader interests, of which the use of economics and mechanism design to make more open, free, egalitarian and efficient systems for human cooperation, including improving or replacing present-day corporations and governments, is a major one. The intersection of interests between the Ethereum community and Posner and Weyl's work is multifaceted and plentiful; _Radical Markets_ dedicates an entire chapter to the idea of "markets for personal data", redefining the economic relationship between ourselves and services like Facebook, and well, look what the Ethereum community is working on: markets for personal data.

Second, blockchains may well be used as a technical backbone for some of the solutions described in the book, and Ethereum-style smart contracts are ideal for the kinds of complex systems of property rights that the book explores. Third, the economic ideas and challenges that the book brings up are ideas that have also been explored, and will be continue to be explored, at great length by the blockchain community for its own purposes. Posner and Weyl's ideas often have the feature that they allow economic incentive alignment to serve as a substitute for subjective ad-hoc bureaucracy (eg. Harberger taxes can essentially replace eminent domain), and given that blockchains lack access to trusted human-controlled courts, these kinds of solutions may prove to be be even more ideal for blockchain-based markets than they are for "real life".

I will warn that readers are not at all guaranteed to find the book's proposals acceptable; at least the first three have already been highly controversial and they do contravene many people's moral preconceptions about how property should and should work and where money and markets can and can't be used. The authors are no strangers to controversy; Posner has on previous occasions even proven willing to argue against such notions as human rights law. That said, the book does go to considerable lengths to explain why each proposal improves efficiency if it could be done, and offer multiple versions of each proposal in the hopes that there is at least one (even if partial) implementation of each idea that any given reader can find agreeable.

What do Posner and Weyl talk about?

The book is split into five major sections, each arguing for a particular reform: self-assessed property taxes, quadratic voting, a new kind of immigration program, breaking up big financial conglomerates that currently make banks and other industries act like monopolies even if they appear at first glance to be competitive, and markets for selling personal data. Properly summarizing all five sections and doing them justice would take too long, so I will focus on a deep summary of one specific section, dealing with a new kind of property taxation, to give the reader a feel for the kinds of ideas that the book is about.

Harberger taxes

See also: "Property Is Only Another Name for Monopoly", Posner and Weyl

Markets and private property are two ideas that are often considered together, and it is difficult in modern discourse to imagine one without (or even with much less of) the other. In the 19th century, however, many economists in Europe were both libertarian _and_ egalitarian, and it was quite common to appreciate markets while maintaining skepticism toward the excesses of private property. A rather

interesting example of this is the [Bastiat-Proudhon debate](#) from 1849-1850 where the two dispute the legitimacy of charging interest on loans, with one side focusing on the mutual gains from voluntary contracts and the other focusing on their suspicion of the potential for people with capital to get even richer without working, leading to unbalanced capital accumulation.

As it turns out, it is absolutely possible to have a system that contains markets but not property rights: at the end of every year, collect every piece of property, and at the start of the next year have the government auction every piece out to the highest bidder. This kind of system is intuitively quite unrealistic and impractical, but it has the benefit that it achieves perfect **allocative efficiency**: every year, every object goes to the person who can derive the most value from it (ie. the highest bidder). It also gives the government a large amount of revenue that could be used to completely substitute income and sales taxes or fund a basic income.

Now you might ask: doesn't the existing property system also achieve allocative efficiency? After all, if I have an apple, and I value it at $2, and you value it at $3, then you could offer me $2.50 and I would accept. However, this fails to take into account imperfect information: how do you know that I value it at $2, and not $2.70? You could offer to buy it for $2.99 so that you can be sure that you'll get it if you really are the one who values the apple more, but then you would be gaining practically nothing from the transaction. And if you ask me to set the price, how do I know that you value it at $3, and not $2.30? And if I set the price to $2.01 to be sure, I would be gaining practically nothing from the transaction. Unfortunately, there is a result known as the [Myerson-Satterthwaite Theorem](#) which means that *no* solution is efficient; that is, any bargaining algorithm in such a situation must at least sometimes lead to inefficiency from mutually beneficial deals falling through.

If there are many buyers you have to negotiate with, things get even harder. If a developer (in the real estate sense) is trying to make a large project that requires buying 100 existing properties, and 99 have already agreed, the remaining one has a strong incentive to charge a very high price, much higher than their actual personal valuation of the property, hoping that the developer will have no choice but to pay up.
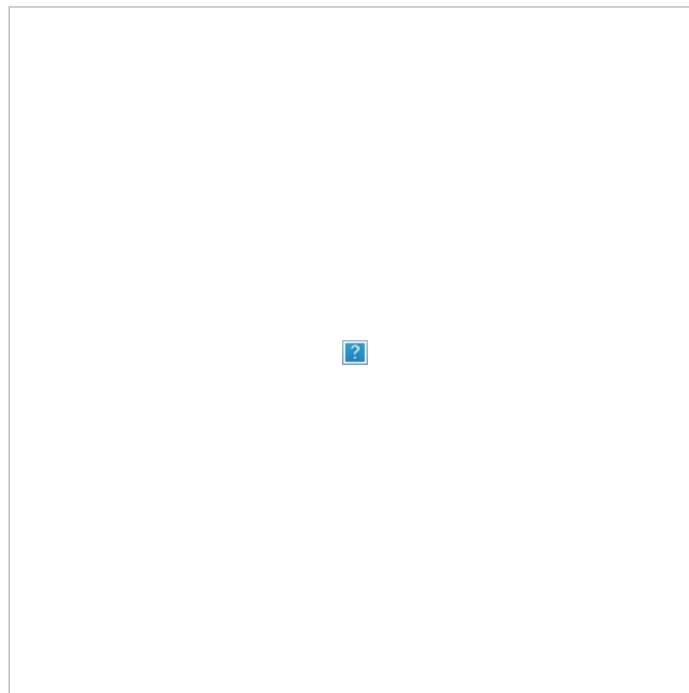
Well, not necessarily no choice. But a very inconvenient and both privately and socially wasteful choice.

Re-auctioning everything once a year completely solves this problem of allocative efficiency, but at a very high cost to **investment efficiency**: there's no point in building a house in the first place if six months later it will get taken away from you and re-sold in an auction. All property taxes have this problem; if building a house costs you $90 and brings you $100 of benefit, but then you have to pay $15 more property tax if you build the house, then you will not build the house and that $10 gain is lost to society.

One of the more interesting ideas from the 19th century economists, and specifically Henry George,

was a kind of property tax that did not have this problem: the [land value tax](). The idea is to charge tax on the value of land, but not the *improvements to the land*; if you own a $100,000 plot of dirt you would have to pay $5,000 per year taxes on it regardless of whether you used the land to build a condominium or simply as a place to walk your pet doge.

A doge.

Weyl and Posner are not convinced that Georgian land taxes are viable in practice:

> Consider, for example, the Empire State Building. What is the pure value of the land beneath it? One could try to infer its value by comparing it to the value of adjoining land. But the building itself defines the neighborhood around it; removing the building would almost certainly change the value of its surrounding land. The land and the building, even the neighborhood, are so tied together, it would be hard to figure out a separate value for each of them.

Arguably this does not exclude the possibility of a different kind of Georgian-style land tax: a tax based on the *average* of property values across a sufficiently large area. That would preserve the property that improving a single piece of land would not (greatly) perversely increase the taxes that they have to pay, without having to find a way to distinguish land from improvements in an absolute sense. But in any case, Posner and Weyl move on to their main proposal: self-assessed property taxes.

Consider a system where property owners themselves specify what the value of their property is, and pay a tax rate of, say, 2% of that value per year. But here is the twist: whatever value they specify for their property, *they have to be willing to sell it to anyone at that price*.

If the tax rate is equal to the chance per year that the property gets sold, then this achieves optimal allocative efficiency: raising your self-assessed property value by $1 increases the tax you pay by $0.02, but it also means there is a 2% chance that someone will buy the property and pay $1 more, so there is no incentive to cheat in either direction. It does harm investment efficiency, but vastly less so than all property being re-auctioned every year.

Posner and Weyl then point out that if more investment efficiency is desired, a hybrid solution with a lower property tax is possible:

> When the tax is reduced incrementally to improve investment efficiency, the loss in allocative efficiency is less than the gain in investment efficiency. The reason is that the most valuable sales are ones where the buyer is willing to pay significantly more than the seller is willing to accept. These transactions are the first ones enabled by a reduction in the price as even a small price reduction will avoid blocking these most valuable transactions. In fact, it can be shown that the size of the social loss from monopoly power grows quadratically in the extent of this power. Thus, reducing the markup by a third eliminates close to $\frac{5}{9} = (3^2-2^2)/(3^2)$ of the allocative harm from private ownership.

This concept of quadratic deadweight loss is a truly important insight in economics, and is arguably the deep reason why "moderation in all things" is such an attractive principle: the first step you take away from an extreme will generally be the most valuable.
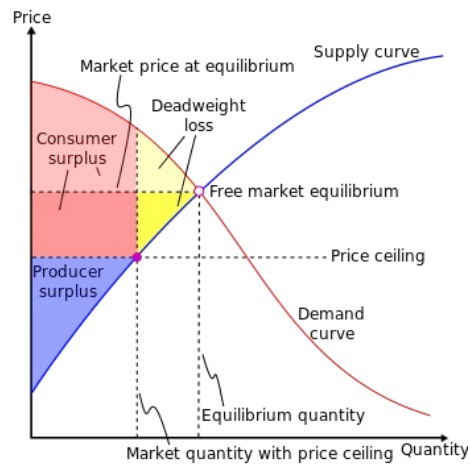
The book then proceeds to give a series of side benefits that this tax would have, as well as some downsides. One interesting side benefit is that it removes an information asymmetry flaw that exists with property sales today, where owners have the incentive to expend effort on making their property look good even in potentially misleading ways. With a properly set Harberger tax, if you somehow mange to trick the world into thinking your house is 5% more valuable, you'll get 5% more when you sell it but until that point you'll have to pay 5% more in taxes, or else someone will much more quickly snap it up from you at the original price.

The downsides are smaller than they seem; for example, one natural disadvantage is that it exposes property owners to uncertainty due to the possibility that someone will snap up their property at any time, but that is hardly an unknown as it's a risk that renters already face every day. But Weyl and Posner *do* propose more moderate ways of introducing the tax that don't have these issues. First, the tax can be applied to types of property that are currently government owned; it's a potentially superior alternative to both continued government ownership *and* traditional full-on privatization. Second, the tax can be applied to forms of property that are already "industrial" in usage: radio spectrum licenses, domain names, intellectual property, etc.

The Rest of the Book

The remaining chapters bring up similar ideas that are similar in spirit to the discussion on Harberger taxes in their use of modern game-theoretic principles to make mathematically optimized versions of existing social institutions. One of the proposals is for something called quadratic voting, which I summarize as follows.

Suppose that you can vote as many times as you want, but voting costs "voting tokens" (say each citizen is assigned N voting tokens per year), and it costs tokens in a nonlinear way: your first vote costs one token, your second vote costs two tokens, and so forth. If someone feels more strongly about something, the argument goes, they would be willing to pay more for a single vote; quadratic voting takes advantage of this by perfectly aligning *quantity* of votes with *cost* of votes: if you're willing to pay up to 15 tokens for a vote, then you will keep buying votes until your last one costs 15 tokens, and so you will cast 15 votes in total. If you're willing to pay up to 30 tokens for a vote, then you will keep buying votes until you can't buy any more for a price less than or equal to 30 tokens, and so you will end up casting 30 votes. The voting is "quadratic" because the total amount you pay for N votes goes up proportionately to N^2.
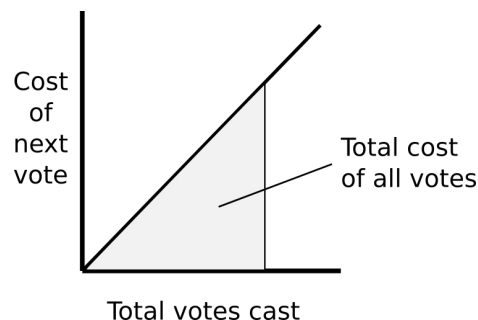
After this, the book describes a market for immigration visas that could greatly expand the number of immigrants admitted while making sure local residents benefit and at the same time aligning incentives to encourage visa sponsors to choose immigrants that are more ikely to succeed in the country and less likely to commit crimes, then an enhancement to antitrust law, and finally the idea of setting up markets for personal data.

Markets in Everything

There are plenty of ways that one could respond to each individual proposal made in the book. I personally, for example, find the immigration visa scheme that Posner and Weyl propose well-intentioned and see how it could improve on the status quo, but also overcomplicated, and it seems simpler to me to have a scheme where visas are auctioned or sold every year, with an additional requirement for migrants to obtain liability insurance. Robin Hanson recently [proposed](#) greatly expanding liability insurance mandates as an alternative to many kinds of regulation, and while imposing new mandates on an entire society seems unrealistic, a new expanded immigration program seems like the perfect place to start considering them. Paying people for personal data is interesting, but there are concerns about adverse selection: to put it politely, the kinds of people that are willing to sit around submitting lots of data to Facebook all year to earn $16.92 (Facebook's current [annualized revenue per user](#)) are *not* the kinds of people that advertisers are willing to burn hundreds of dollars per person trying to market rolexes and Lambos to. However, what I find more interesting is the general principle that the book tries to promote.

Over the last hundred years, there truly has been a large amount of research into designing economic mechanisms that have desirable properties and that outperform simple two-sided buy-and-sell markets. Some of this research has been put into use in some specific industries; for example, [combinatorial auctions](#) are used in airports, radio spectrum auctions and several other industrial use cases, but it hasn't really seeped into any kind of broader policy design; the political systems and property rights that we have are still largely the same as we had two centuries ago. So can we use modern economic insights to reform base-layer markets and politics in such a deep way, and if so, should we?

Normally, I love markets and clean incentive alignment, and dislike politics and bureaucrats and ugly hacks, and I love economics, and I so love the idea of using economic insights to design markets that work better so that we can reduce the role of politics and bureaucrats and ugly hacks in society. Hence, naturally, I love this vision. So let me be a good intellectual citizen and do my best to try to make a case against it.

There is a limit to how complex economic incentive structures and markets can be because there is a limit to users' ability to think and re-evaluate and give ongoing precise measurements for their valuations of things, and people value reliability and certainty. Quoting [Steve Waldman criticizing Uber surge pricing](#):

> Finally, we need to consider questions of economic calculation. In macroeconomics, we sometimes face tradeoffs between an increasing and unpredictably variable price-level and full employment. Wisely or not, our current policy is to stabilize the price level, even at short-term cost to output and employment, because stable prices enable longer-term economic calculation. That vague good, not visible on a supply/demand diagram, is deemed worth very large sacrifices. The same concern exists in a microeconomic context. If the "ride-sharing revolution" really takes hold, a lot of us will have decisions to make about whether to own a car or rely upon the Sidecars, Lyfts, and Ubers of the world to take us to work every day. To make those calculations, we will need something like predictable pricing. Commuting to our minimum wage jobs (average is over!) by Uber may be OK at standard pricing, but not so OK on a surge. In the desperate utopia of the "free-market economist", there is always a solution to this problem. We can define futures markets on Uber trips, and so hedge our exposure to price volatility! In practice that is not so likely...

And:

> It's clear that in a lot of contexts, people have a strong preference for price-predictability over immediate access. The vast majority of services that we purchase and consume are not price-rationed in any fine-grained way. If your hairdresser or auto mechanic is busy, you get penciled in for next week...

Strong property rights are valuable for the same reason: beyond the arguments about allocative and investment efficiency, they provide the mental convenience and planning benefits of predictability.

It's worth noting that even Uber itself doesn't do surge pricing in the "market-based" way that economists would recommend. Uber is not a market where drivers can set their own prices, riders

can see what prices are available, and themselves choose their tradeoff between price and waiting time. Why does Uber not do this? One argument is that, as Steve Waldman says, "Uber itself is a cartel", and wants to have the power to adjust market prices not just for efficiency but also reasons such as profit maximization, strategically setting prices to drive out competing platforms (and taxis and public transit), and public relations. As Waldman further points out, one Uber competitor, Sidecar, *does* have the ability for [drivers to set prices](), and I would add that I have seen ride-sharing apps in China where *passengers* can offer drivers higher prices to try to coax them to get a car faster.

A possible counter-argument that Uber might give is that drivers themselves are actually less good at setting optimal prices than Uber's own algorithms, and in general people value the convenience of one-click interfaces over the mental complexity of thinking about prices. If we assume that Uber won its market dominance over competitors like Sidecar fairly, then the market itself has decided that the economic gain from marketizing more things is not worth the mental transaction costs.

Harberger taxes, at least to me, seem like they would lead to these exact kinds of issues multipled by ten; people are not experts at property valuation, and would have to spend a significant amount of time and mental effort figuring out what self-assessed value to put for their house, and they would complain much more if they accidentally put a value that's too low and suddenly find that their house is gone. If Harberger taxes were to be applied to smaller property items as well, people would need to juggle a large amount of mental valuations of everything. A similar critique could apply to many kinds of personal data markets, and possibly even to quadratic voting if implemented in its full form.

I could challenge this by saying "ah, even if that's true, this is the 21st century, we could have companies that build AIs that make pricing decisions on your behalf, and people could choose the AI that seems to work best; there could even be a public option"; and Posner and Weyl themselves suggest that this is likely the way to go. And this is where the interesting conversation starts.

Tales from Crypto Land

One reason why this discussion particularly interests me is that the cryptocurrency and blockchain space itself has, in some cases, run up against similar challenges. In the case of Harberger taxes, we actually did consider almost exactly that same proposal in the context of the [Ethereum Name System]() (our decentralized alternative to DNS), but the proposal was ultimately rejected. I asked the ENS developers why it was rejected. Paraphrasing their reply, the challenge is as follows.

Many ENS domain names are of a type that would only be interesting to precisely two classes of actors: (i) the "legitimate owner" of some given name, and (ii) scammers. Furthermore, in some particular cases, the legitimate owner is uniquely underfunded, and scammers are uniquely dangerous. One particular case is [MyEtherWallet](), an Ethereum wallet provider. MyEtherWallet provides an important public good to the Ethereum ecosystem, making Ethereum easier to use for many thousands of people, but is able to capture only a very small portion of the value that it provides; as a result, the budget that it has for outbidding others for the domain name is low. If a scammer gets their hands on the domain, users trusting MyEtherWallet could easily be tricked into sending all of their ether (or other Ethereum assets) to a scammer. Hence, because there is generally one clear "legitimate owner" for any domain name, a pure property rights regime presents little allocative efficiency loss, and there is a strong overriding public interest toward stability of reference (ie. a domain that's legitimate one day doesn't redirect to a scam the next day), so *any* level of Harberger taxation may well bring more harm than good.

I suggested to the ENS developers the idea of applying Harberger taxes to short domains (eg. abc.eth), but not long ones; the reply was that it would be too complicated to have two classes of names. That said, perhaps there is some version of the proposal that could satisfy the specific constraints here; I would be interested to hear Posner and Weyl's feedback on this particular application.

Another story from the blockchain and Ethereum space that has a more pro-radical-market conclusion is that of transaction fees. The notion of [mental transaction costs](), the idea that the inconvenience of even thinking about whether or not some small payment for a given digital good is worth it is enough of a burden to prevent "micro-markets" from working, is often used as an argument for why mass adoption of blockchain tech would be difficult: every transaction requires a small fee, and the mental expenditure of figuring out what fee to pay is itself a major usability barrier. These arguments increased further at the end of last year, when both [Bitcoin]() and [Ethereum]() transaction fees briefly spiked up by a factor of over 100 due to high usage (talk about surge pricing!), and those who accidentally did not pay high enough fees saw their transactions get stuck for days.

That said, this is a problem that we have now, arguably, to a large extent overcome. After the spikes at the end of last year, Ethereum wallets developed more advanced algorithms for choosing what transaction fees to pay to ensure that one's transaction gets included in the chain, and today most users are happy to simply defer to them. In my own personal experience, the mental transaction costs of worrying about transaction fees do not really exist, much like a driver of a car does not worry about the gasoline consumed by every single turn, acceleration and braking made by their car.
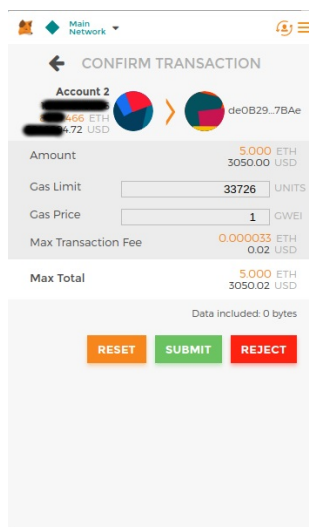
Personal price-setting AIs for interacting with open markets: already a reality in the Ethereum transaction fee market

A third kind of "radical market" that we are considering implementing in the context of Ethereum's consensus system is one for incentivizing deconcentration of validator nodes in [proof of stake consensus](). It's important for blockchains to be decentralized, a similar challenge to what antitrust law tries to solve, but the tools at our disposal are different. Posner and Weyl's solution to antitrust, banning institutional investment funds from owning shares in multiple competitors in the same industry, is far too subjective and human-judgement-dependent to work in a blockchain, but for our specific context we have a different solution: if a validator node commits an error, it gets penalized an amount proportional to the number of other nodes that have committed an error around the same time. This incentivizes nodes to set themselves up in such a way that their failure rate is maximally uncorrelated with everyone else's failure rate, reducing the chance that many nodes fail at the same time and threaten to the blockchain's integrity. I want to ask Posner and Weyl: though our exact approach is fairly application-specific, could a similarly elegant "market-based" solution be discovered to incentivize market deconcentration in general?

All in all, I am optimistic that the various behavioral kinks around implementing "radical markets" in practice could be worked out with the help of good defaults and personal AIs, though I do think that if this vision is to be pushed forward, the greatest challenge will be finding progressively larger and more meaningful places to test it out and show that the model works. I particularly welcome the use of the blockchain and crypto space as a testing ground.

Another Kind of Radical Market

The book as a whole tends to focus on centralized reforms that could be implemented on an economy from the top down, even if their intended long-term effect is to push more decision-making power to individuals. The proposals involve large-scale restructurings of how property rights work, how voting works, how immigration and antitrust law works, and how individuals see their relationship with property, money, prices and society. But there is also the potential to use economics and game theory to come up with *decentralized* economic institutions that could be adopted by smaller groups of people at a time.

Perhaps the most famous examples of decentralized institutions from game theory and economics land are (i) assurance contracts, and (ii) prediction markets. An assurance contract is a system where some public good is funded by giving anyone the opportunity to pledge money, and only collecting the pledges if the total amount pledged exceeds some threshold. This ensures that people can donate money knowing that either they will get their money back or there actually will be enough to achieve

some objective. A possible extension of this concept is Alex Tabarrok's [dominant assurance contracts](#), where an entrepreneur offers to refund participants *more* than 100% of their deposits if a given assurance contract does not raise enough money.

Prediction markets allow people to bet on the probability that events will happen, potentially even conditional on some action being taken ("I bet $20 that unemployment will go down if candidate X wins the election"); there are techniques for people interested in the information to subsidize the markets. Any attempt to manipulate the probability that a prediction market shows simply creates an opportunity for people to earn free money (yes I know, risk aversion and capital efficiency etc etc; still close to free) by betting against the manipulator.

Posner and Weyl do give one example of what I would call a decentralized institution: a game for choosing who gets an asset in the event of a divorce or a company splitting in half, where both sides provide their own valuation, the person with the higher valuation gets the item, but they must then give an amount equal to half the average of the two valuations to the loser. There's some economic reasoning by which this solution, while not perfect, is still close to mathematically optimal.

One particular category of decentralized institutions I've been interested in is improving incentivization for content posting and content curation in social media. Some ideas that I have had include:

- [Proof of stake conditional hashcash](#) (when you send someone an email, you give them the opportunity to burn $0.5 of your money if they think it's spam)
- [Prediction markets for content curation](#) (use prediction markets to predict the results of a moderation vote on content, thereby encouraging a market of fast content pre-moderators while penalizing manipulative pre-moderation)
- Conditional payments for paywalled content (after you pay for a piece of downloadable content and view it, you can decide after the fact if payments should go to the author or to proportionately refund previous readers)

And ideas I have had in other contexts:

- [Call-out assurance contracts](#)
- [DAICOs](#) (a more decentralized and safer alternative to ICOs)
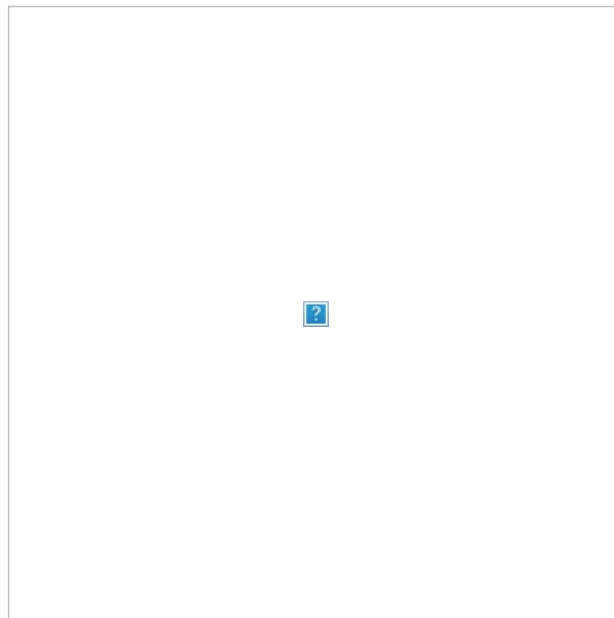
Twitter scammers: can prediction markets incentivize an autonomous swarm of human and AI-driven moderators to flag these posts and warn users not to send them ether within a few seconds of the post being made? And could such a system be generalized to the entire internet, where these is no single centralized moderator that can easily take posts down?

Some ideas others have had for decentralized institutions in general include:

- [TrustDavis](#) (adding skin-in-the-game to e-commerce reputations by making e-commerce ratings

be offers to insure others against the receiver of the rating committing fraud)
- [Circles](#) (decentralized basic income through locally fungible coin issuance)
- Markets for CAPTCHA services
- Digitized peer to peer rotating savings and credit [associations](#)
- [Token curated registries](#)
- [Crowdsourced smart contract truth oracles](#)
- Using blockchain-based smart contracts to coordinate unions

I would be interested in hearing Posner and Weyl's opinion on these kinds of "radical markets", that groups of people can spin up and start using by themselves without requiring potentially contentious society-wide changes to political and property rights. Could decentralized institutions like these be used to solve the key defining challenges of the twenty first century: promoting beneficial scientific progress, developing informational public goods, reducing global wealth inequality, and the big meta-problem behind fake news, government-driven and corporate-driven social media censorship, and regulation of cryptocurrency products: how do we do quality assurance in an open society?

All in all, I highly recommend *Radical Markets* (and by the way I also recommend Eliezer Yudkowsky's *[Inadequate Equilibria](#)*) to anyone interested in these kinds of issues, and look forward to seeing the discussion that the book generates.

Governance, Part 2: Plutocracy Is Still Bad

2018 Mar 28

Coin holder voting, both for governance of technical features, and for more extensive use cases like deciding who runs validator nodes and who receives money from development bounty funds, is unfortunately continuing to be popular, and so it seems worthwhile for me to write another post explaining why I (and Vlad Zamfir and others) do not consider it wise for Ethereum (or really, any base-layer blockchain) to start adopting these kinds of mechanisms in a tightly coupled form in any significant way.

I wrote about the issues with tightly coupled voting in a blog post last year, that focused on theoretical issues as well as focusing on some practical issues experienced by voting systems over the previous two years. Now, the latest scandal in DPOS land seems to be substantially worse. Because the delegate rewards in EOS are now so high (5% annual inflation, about $400m per year), the competition on who gets to run nodes has essentially become yet another frontier of US-China geopolitical economic warfare.

And that's not my own interpretation; I quote from this article (original in Chinese):

> **EOS supernode voting: multibillion-dollar profits leading to crypto community inter-country warfare**
>
> Looking at community recognition, Chinese nodes feel much less represented in the community than US and Korea. Since the EOS.IO official Twitter account was founded, there has never been any interaction with the mainland Chinese EOS community. For a listing of the EOS officially promoted events and interactions with communities see the picture below.
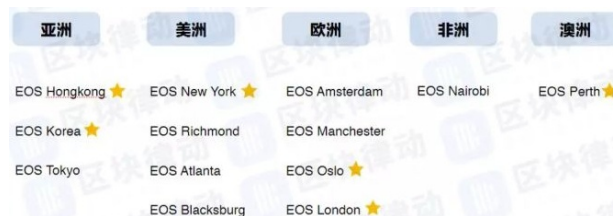
> With no support from the developer community, facing competition from Korea, the Chinese EOS supernodes have invented a new strategy: buying votes.

The article then continues to describe further strategies, like forming "alliances" that all vote (or buy votes) for each other.

Of course, it does not matter at all who the specific actors are that are buying votes or forming cartels; this time it's some Chinese pools, last time it was "members located in the USA, Russia, India, Germany, Canada, Italy, Portugal and many other countries from around the globe", next time

it could be totally anonymous, or run out of a smartphone snuck into Trendon Shavers's prison cell. What matters is that blockchains and cryptocurrency, originally founded in a vision of using technology to escape from the failures of human politics, have essentially all but replicated it. Crypto is a reflection of the world at large.

The EOS New York community's response seems to be that they have issued a strongly worded letter to the world stating that buying votes will be against the constitution. Hmm, what other major political entity has made accepting bribes a violation of the constitution? And how has that been going for them lately?

The second part of this article will involve me, an armchair economist, hopefully convincing you, the reader, that yes, bribery is, in fact, bad. There are actually people who dispute this claim; the usual argument has something to do with market efficiency, as in "isn't this good, because it means that the nodes that win will be the nodes that can be the cheapest, taking the least money for themselves and their expenses and giving the rest back to the community?" The answer is, kinda yes, but in a way that's centralizing and vulnerable to rent-seeking cartels and explicitly contradicts many of the explicit promises made by most DPOS proponents along the way.

Let us create a toy economic model as follows. There are a number of people all of which are running to be delegates. The delegate slot gives a reward of $100 per period, and candidates promise to share some portion of that as a bribe, equally split among all of their voters. The actual N delegates (eg. $N = 35$) in any period are the N delegates that received the most votes; that is, during every period a threshold of votes emerges where if you get more votes than that threshold you are a delegate, if you get less you are not, and the threshold is set so that N delegates are above the threshold.

We expect that voters vote for the candidate that gives them the highest expected bribe. Suppose that all candidates start off by sharing 1%; that is, equally splitting $1 among all of their voters. Then, if some candidate becomes a delegate with K voters, each voter gets a payment of $\frac{1}{K}$. The candidate that it's most profitable to vote for is a candidate that's expected to be in the top N, but is expected to earn the fewest votes within that set. Thus, we expect votes to be fairly evenly split among 35 delegates.

Now, some candidates will want to secure their position by sharing more; by sharing 2%, you are likely to get twice as many votes as those that share 1%, as that's the equilibrium point where voting for you has the same payout as voting for anyone else. The extra guarantee of being elected that this gives is definitely worth losing an additional 1% of your revenue when you do get elected. We can expect delegates to bid up their bribes and eventually share something close to 100% of their revenue. So the outcome seems to be that the delegate payouts are largely simply returned to voters, making the delegate payout mechanism close to meaningless.

But it gets worse. At this point, there's an incentive for delegates to form alliances (aka political parties, aka cartels) to coordinate their share percentages; this reduces losses to the cartel from chaotic competition that accidentally leads to some delegates not getting enough votes. Once a cartel is in place, it can start bringing its share percentages down, as dislodging it is a hard coordination problem: if a cartel offers 80%, then a new entrant offers 90%, then to a voter, seeking a share of that extra 10% is not worth the risk of either (i) voting for someone who gets insufficient votes and does not pay rewards, or (ii) voting for someone who gets too many votes and so pays out a reward that's excessively diluted.

Return to voters
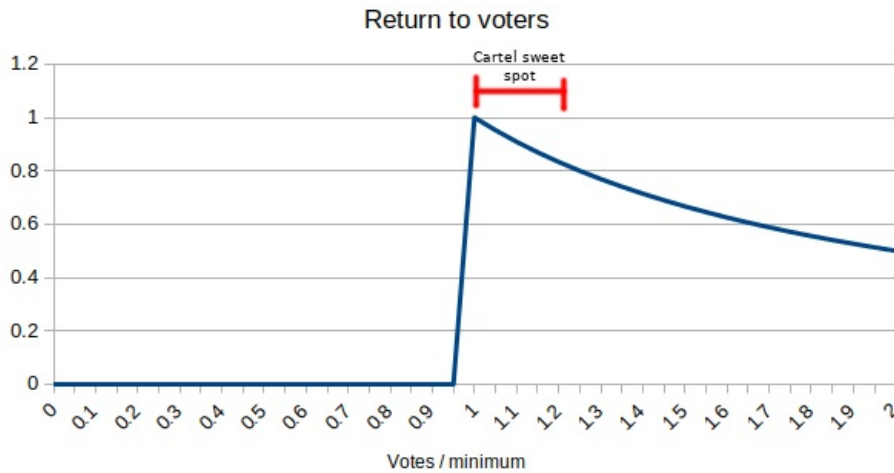
Sidenote: [Bitshares DPOS](#) used approval voting, where you can vote for as many candidates as you want; it should be pretty obvious that with even slight bribery, the equilibrium there is that everyone just votes for everyone.

Furthermore, even if cartel mechanics *don't* come into play, there is a further issue. This equilibrium of coin holders voting for whoever gives them the most bribes, or a cartel that has become an entrenched rent seeker, contradicts explicit promises made by DPOS proponents.

Quoting "[Explain Delegated Proof of Stake Like I'm 5](#)":

> If a Witness starts acting like an asshole, or stops doing a quality job securing the network, people in the community can remove their votes, essentially firing the bad actor. Voting is always ongoing.

From "[EOS: An Introduction](#)":

> By custom, we suggest that the bulk of the value be returned to the community for the common good - software improvements, dispute resolution, and the like can be entertained. In the spirit of "eating our own dogfood," the design envisages that the community votes on a set of open entry contracts that act like "foundations" for the benefit of the community. Known as Community Benefit Contracts, the mechanism highlights the importance of DPOS as enabling direct on-chain governance by the community (below).

The flaw in all of this, of course, is that the average voter has only a very small chance of impacting which delegates get selected, and so they only have a very small incentive to vote based on any of these high-minded and lofty goals; rather, their incentive is to vote for whoever offers the highest and most reliable bribe. Attacking is easy. If a cartel equilibrium does not form, then an attacker can simply offer a share percentage slightly higher than 100% (perhaps using fee sharing or some kind of "starter promotion" as justification), capture the majority of delegate positions, and then start an attack. If they get removed from the delegate position via a hard fork, they can simply restart the attack again with a different identity.

The above is not intended purely as a criticism of DPOS consensus or its use in any specific blockchain. Rather, the critique reaches much further. There has been a large number of projects recently that extol the virtues of extensive on-chain governance, where on-chain coin holder voting can be used not just to vote on protocol features, but also to control a bounty fund. Quoting a [blog post from last year](#):

> Anyone can submit a change to the governance structure in the form of a code update. An on-chain vote occurs, and if passed, the update makes its way on to a test network. After a period of time on the test network, a confirmation vote occurs, at which point the change goes live on the main network. They call this concept a "self-amending ledger". Such a system is interesting because it shifts power towards users and away from the more centralized group of developers and miners. On the developer side, anyone can submit a change, and most importantly, everyone has an economic incentive to do it. Contributions are rewarded by the community with newly minted tokens through inflation funding. This

shifts from the current Bitcoin and Ethereum dynamics where a new developer has little incentive to evolve the protocol, thus power tends to concentrate amongst the existing developers, to one where everyone has equal earning power.

In practice, of course, what this can easily lead to is funds that offer kickbacks to users who vote for them, leading to the exact scenario that we saw above with DPOS delegates. In the best case, the funds will simply be returned to voters, giving coin holders an interest rate that cancels out the inflation, and in the worst case, some portion of the inflation will get captured as economic rent by a cartel.

Note also that the above is not a criticism of *all* on-chain voting; it does not rule out systems like futarchy. However, futarchy is untested, but coin voting *is* tested, and so far it seems to lead to a high risk of economic or political failure of some kind - far too high a risk for a platform that seeks to be an economic base layer for development of decentralized applications and institutions.

So what's the alternative? The answer is what we've been saying all along: *cryptoeconomics*. Cryptoeconomics is fundamentally about the use of economic incentives together with cryptography to design and secure different kinds of systems and applications, including consensus protocols. The goal is simple: to be able to measure the security of a system (that is, the cost of breaking the system or causing it to violate certain guarantees) in dollars. Traditionally, the security of systems often depends on *social* trust assumptions: the system works if 2 of 3 of Alice, Bob and Charlie are honest, and we trust Alice, Bob and Charlie to be honest because I know Alice and she's a nice girl, Bob registered with FINCEN and has a money transmitter license, and Charlie has run a successful business for three years and wears a suit.

Social trust assumptions can work well in many contexts, but they are difficult to universalize; what is trusted in one country or one company or one political tribe may not be trusted in others. They are also difficult to quantify; how much money does it take to manipulate social media to favor some particular delegate in a vote? Social trust assumptions seem secure and controllable, in the sense that "people" are in charge, but in reality they can be manipulated by economic incentives in all sorts of ways.

Cryptoeconomics is about trying to reduce social trust assumptions by creating systems where we introduce explicit economic incentives for good behavior and economic penalties for bad behavior, and making mathematical proofs of the form "in order for guarantee X to be violated, at least these people need to misbehave in this way, which means the minimum amount of penalties or foregone revenue that the participants suffer is Y". Casper is designed to accomplish precisely this objective in the context of proof of stake consensus. Yes, this does mean that you can't create a "blockchain" by concentrating the consensus validation into 20 uber-powerful "supernodes" and you have to actually think to make a design that intelligently breaks through and navigates existing tradeoffs and achieves massive scalability in a still-decentralized network. But the reward is that you don't get a network that's constantly liable to breaking in half or becoming economically captured by unpredictable political forces.

---

1. *It has been brought to my attention that EOS may be reducing its delegate rewards from 5% per year to 1% per year. Needless to say, this doesn't really change the fundamental validity of any of the arguments; the only result of this would be 5x less rent extraction potential at the cost of a 5x reduction to the cost of attacking the system.*
2. *Some have asked: but how can it be wrong for DPOS delegates to bribe voters, when it is perfectly legitimate for mining and stake pools to give 99% of their revenues back to their participants? The answer should be clear: in PoW and PoS, it's the protocol's role to determine the rewards that miners and validators get, based on the miners and validators' observed performance, and the fact that miners and validators that are pools pass along the rewards (and penalties!) to their participants gives the participants an incentive to participate in good pools. In DPOS, the reward is constant, and it's the voters' role to vote for pools that have good performance, but with the key flaw that there is no mechanism to actually encourage voters to vote in that way instead of just voting for whoever gives them the most money without taking performance into account. Penalties in DPOS do not exist, and are certainly not passed on to voters, so voters have no "skin in the game" (penalties in Casper pools, on the other hand, **do** get passed on to participants).*

Proof of Stake FAQ

2017 Dec 31

Contents

What is Proof of Stake

Proof of Stake (PoS) is a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network. In proof of work (PoW) based public blockchains (e.g. Bitcoin and the current implementation of Ethereum), the algorithm rewards participants who solve cryptographic puzzles in order to validate transactions and create new blocks (i.e. mining). In PoS-based public blockchains (e.g. Ethereum's upcoming Casper implementation), a set of validators take turns proposing and voting on the next block, and the weight of each validator's vote depends on the size of its deposit (i.e. stake). Significant advantages of PoS include **security, reduced risk of centralization, and energy efficiency**.

In general, a proof of stake algorithm looks as follows. The blockchain keeps track of a set of validators, and anyone who holds the blockchain's base cryptocurrency (in Ethereum's case, ether) can become a validator by sending a special type of transaction that **locks up their ether into a deposit**. The process of creating and agreeing to new blocks is then done through a consensus algorithm that all current validators can participate in.

There are many kinds of consensus algorithms, and many ways to assign rewards to validators who participate in the consensus algorithm, so there are many "flavors" of proof of stake. From an algorithmic perspective, there are two major types: chain-based proof of stake and BFT-style proof of stake.

In **chain-based proof of stake**, the algorithm pseudo-randomly selects a validator during each time slot (e.g. every period of 10 seconds might be a time slot), and assigns that validator the right to create a single block, and this block must point to some previous block (normally the block at the end of the previously longest chain), and so over time most blocks converge into a single constantly growing chain.

In **BFT-style proof of stake**, validators are **randomly** assigned the right to *propose* blocks, but *agreeing on which block is canonical* is done through a multi-round process where every validator sends a "vote" for some specific block during each round, and at the end of the process all (honest and online) validators permanently agree on whether or not any given block is part of the chain. Note that blocks may still be *chained together*; the key difference is that consensus on a block can come within one block, and does not depend on the length or size of the chain after it.

What are the benefits of proof of stake as opposed to proof of work?

See A Proof of Stake Design Philosophy for a more long-form argument.

In short:

- **No need to consume large quantities of electricity** in order to secure a blockchain (e.g. it's estimated that both Bitcoin and Ethereum burn over $1 million worth of electricity and hardware costs per day as part of their consensus mechanism).
- Because of the lack of high electricity consumption, there is **not as much need to issue as many new coins** in order to motivate participants to keep participating in the network. It may theoretically even be possible to have *negative* net issuance, where a portion of transaction fees is "burned" and so the supply goes down over time.
- Proof of stake opens the door to a wider array of techniques that use game-theoretic mechanism design in order to better **discourage centralized cartels** from forming and, if they do form, from acting in ways that are harmful to the network (e.g. like selfish mining in proof of work).
- **Reduced centralization risks**, as economies of scale are much less of an issue. $10 million of coins will get you exactly 10 times higher returns than $1 million of coins, without any additional disproportionate gains because at the higher level you can afford better mass-production equipment, which is an advantage for Proof-of-Work.
- Ability to use economic penalties to **make various forms of 51% attacks vastly more expensive** to carry out than proof of work - to paraphrase Vlad Zamfir, "it's as though your ASIC farm burned down if you participated in a 51% attack".

How does proof of stake fit into traditional Byzantine fault tolerance research?

There are several fundamental results from Byzantine fault tolerance research that apply to all consensus algorithms, including traditional consensus algorithms like PBFT but also any proof of stake algorithm and, with the appropriate mathematical modeling, proof of work.

The key results include:

- **CAP theorem** - "in the cases that a network partition takes place, you have to choose either consistency or availability, you cannot have both". The intuitive argument is simple: if the network splits in half, and in one half I send a transaction "send my 10 coins to A" and in the other I send a transaction "send my 10 coins to B", then either the system is unavailable, as one or both transactions will not be processed, or it becomes inconsistent, as one half of the network will see the first transaction completed and the other half will see the second transaction completed. Note that the CAP theorem has nothing to do with scalability; it applies to sharded and non-sharded systems equally. See also https://github.com/ethereum/wiki/wiki/Sharding-FAQs#but-doesnt-the-cap-theorem-mean-that-fully-secure-distributed-systems-are-impossible-and-so-sharding-is-futile.
- **FLP impossibility** - in an asynchronous setting (i.e. there are no guaranteed bounds on network latency even between correctly functioning nodes), it is not possible to create an algorithm which is guaranteed to reach consensus in any specific finite amount of time if even a single faulty/dishonest node is present. Note that this does NOT rule out "Las Vegas" algorithms that have some probability each round of achieving consensus and thus will achieve consensus within T seconds with probability exponentially approaching 1 as T grows; this is in fact the "escape hatch" that many successful consensus algorithms use.
- **Bounds on fault tolerance** - from the DLS paper we have: (i) protocols running in a partially synchronous network model (i.e. there is a bound on network latency but we do not know ahead of time what it is) can tolerate up to 1/3 arbitrary (i.e. "Byzantine") faults, (ii) deterministic protocols in an asynchronous model (i.e. no bounds on network latency) cannot tolerate faults (although their paper fails to mention that randomized algorithms can with up to 1/3 fault tolerance), (iii) protocols in a synchronous model (i.e. network latency is guaranteed to be less than a known *d*) can, surprisingly, tolerate up to 100% fault tolerance, although there are restrictions on what can happen when more than or equal to 1/2 of nodes are faulty. Note that the "authenticated Byzantine" model is the one worth considering, not the "Byzantine" one; the "authenticated" part essentially means that we can use public key cryptography in our algorithms, which is in modern times very well-researched and very cheap.

Proof of work has been rigorously analyzed by Andrew Miller and others and fits into the picture as an algorithm reliant on a synchronous network model. We can model the network as being made up of a near-infinite number of nodes, with each node representing a very small unit of computing power and having a very small probability of being able to create a block in a given period. In this model, the protocol has 50% fault tolerance assuming zero network latency, ~46% (Ethereum) and ~49.5% (Bitcoin) fault tolerance under actually observed conditions, but goes down to 33% if network latency is equal to the block time, and reduces to zero as network latency approaches infinity.

Proof of stake consensus fits more directly into the Byzantine fault tolerant consensus mould, as all validators have known identities (stable Ethereum addresses) and the network keeps track of the total size of the validator set. There are two general lines of proof of stake research, one looking at synchronous network models and one looking at partially asynchronous network models. "Chain-based" proof of stake algorithms almost always rely on synchronous network models, and their security can be formally proven within these models similarly to how security of [proof of work algorithms](#) can be proven. A line of research connecting traditional Byzantine fault tolerant consensus in partially synchronous networks to proof of stake also exists, but is more complex to explain; it will be covered in more detail in later sections.

Proof of work algorithms and chain-based proof of stake algorithms choose availability over consistency, but BFT-style consensus algorithms lean more toward consistency; [Tendermint](#) chooses consistency explicitly, and Casper uses a hybrid model that prefers availability but provides as much consistency as possible and makes both on-chain applications and clients aware of how strong the consistency guarantee is at any given time.

Note that Ittay Eyal and Emin Gun Sirer's [selfish mining](#) discovery, which places 25% and 33% bounds on the incentive compatibility of Bitcoin mining depending on the network model (i.e. mining is only incentive compatible if collusions larger than 25% or 33% are impossible) has NOTHING to do with results from traditional consensus algorithm research, which does not touch incentive compatibility.

What is the "nothing at stake" problem and how can it be fixed?

In many early (all chain-based) proof of stake algorithms, including Peercoin, there are only rewards for producing blocks, and no penalties. This has the unfortunate consequence that, in the case that there are multiple competing chains, it is in a validator's incentive to try to make blocks on top of every chain at once, just to be sure:
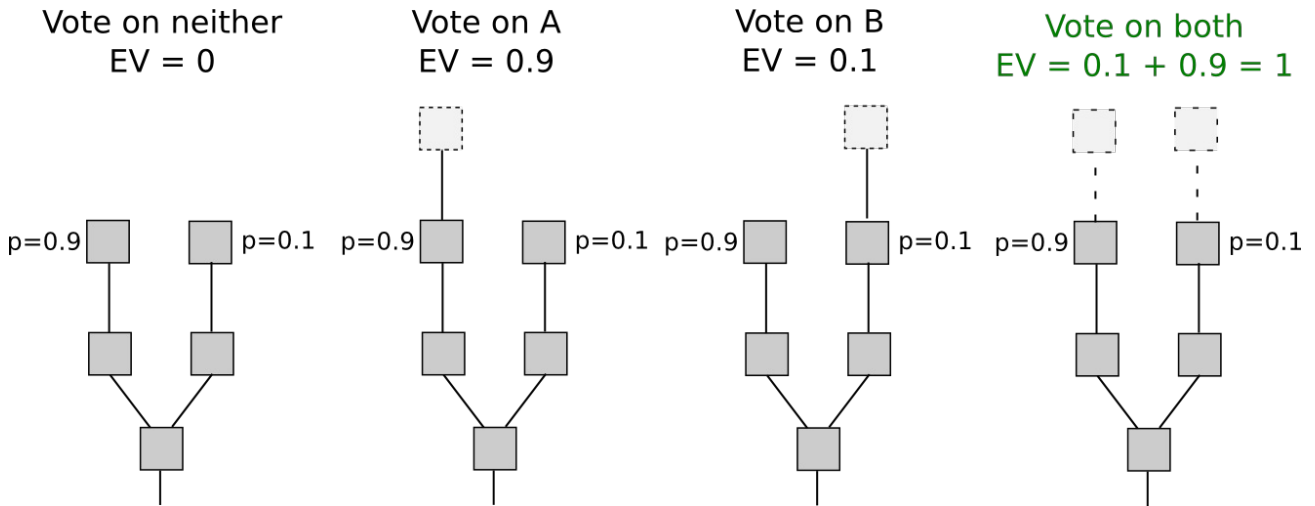
In proof of work, doing so would require splitting one's computing power in half, and so would not be lucrative:
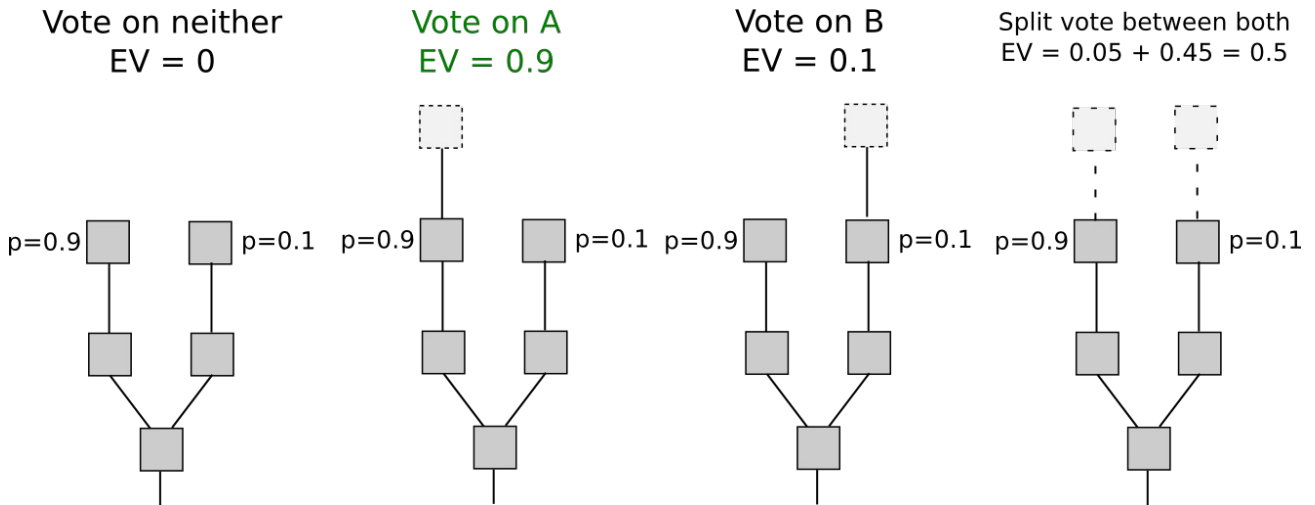
The result is that if all actors are narrowly economically rational, then even if there are no attackers, a blockchain may never reach consensus. If there is an attacker, then the attacker need only overpower altruistic nodes (who would exclusively stake on the original chain), and not rational nodes (who would stake on both the original chain and the attacker's chain), in contrast to proof of work, where the attacker must overpower both altruists and rational nodes (or at least credibly threaten to: see [P + epsilon attacks](#)).

Some argue that stakeholders have an incentive to act correctly and only stake on the longest chain in order to "preserve the value of their investment", however this ignores that this incentive suffers from [tragedy of the commons](#) problems: each individual stakeholder might only have a 1% chance of being "pivotal" (i.e. being in a situation where if they participate in an attack then it succeeds and if they do not participate it fails), and so the bribe needed to convince them personally to join an attack would be only 1% of the size of their deposit; hence, the required combined bribe would be only 0.5-1% of the total sum of all deposits. Additionally, this argument implies that any zero-chance-of-failure situation is not a stable equilibrium, as if the chance of failure is zero then everyone has a 0% chance of being pivotal.

This can be solved via two strategies. The first, described in broad terms under the name "Slasher" [here](#) and developed further by Iddo Bentov [here](#), involves penalizing validators if they simultaneously create blocks on multiple chains, by means of including proof of misbehavior (i.e. two conflicting signed block headers) into the blockchain as a later point in time at which point the malfeasant validator's deposit is deducted appropriately. This changes the incentive structure thus:

| Vote on neither
EV = 0 | Vote on A
EV = 0.9 | Vote on B
EV = 0.1 | Vote on both
EV = 0.1 + 0.9 - 5 = -4 |

$p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$
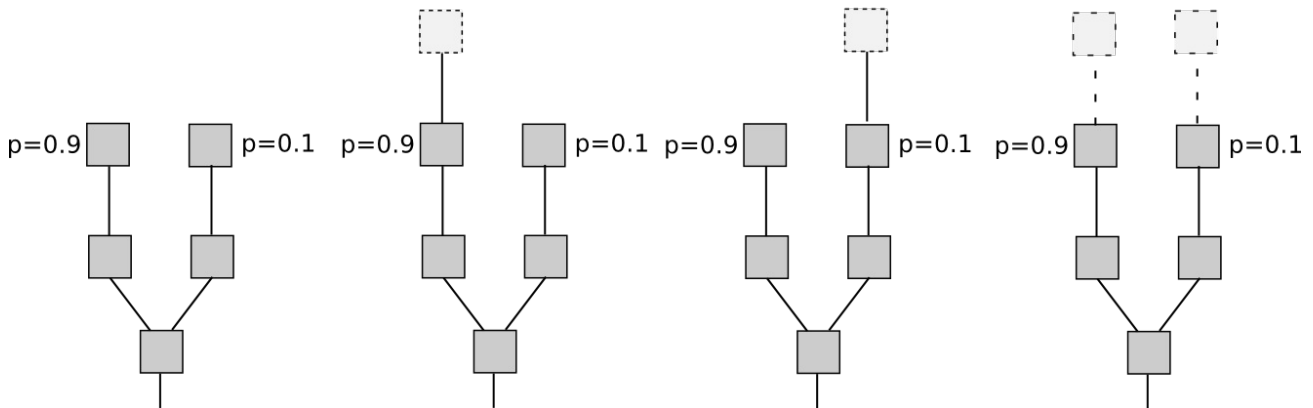
Note that for this algorithm to work, the validator set needs to be determined well ahead of time. Otherwise, if a validator has 1% of the stake, then if there are two branches A and B then 0.99% of the time the validator will be eligible to stake only on A and not on B, 0.99% of the time the validator will be eligible to stake on B and not on A, and only 0.01% of the time will the validator will be eligible to stake on both. Hence, the validator can with 99% efficiency probabilistically double-stake: stake on A if possible, stake on B if possible, and only if the choice between both is open stake on the longer chain. This can only be avoided if the validator selection is the same for every block on both branches, which requires the validators to be selected at a time before the fork takes place.

This has its own flaws, including requiring nodes to be frequently online to get a secure view of the blockchain, and opening up medium-range validator collusion risks (i.e. situations where, for example, 25 out of 30 consecutive validators get together and agree ahead of time to implement a 51% attack on the previous 19 blocks), but if these risks are deemed acceptable then it works well.

The second strategy is to simply punish validators for creating blocks on the *wrong* chain. That is, if there are two competing chains, A and B, then if a validator creates a block on B, they get a reward of +R on B, but the block header can be included into A (in Casper this is called a "dunkle") and on A the validator suffers a penalty of -F (possibly F = R). This changes the economic calculation thus:

| Vote on neither
EV = 0 | Vote on A
EV = 0.9 | Vote on B
EV = 0.1 - 0.9 * 5 = -4.4 | Vote on both
EV = 0.1 + 0.9 - 5 = -4 |

$p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$ $p=0.9$ $p=0.1$
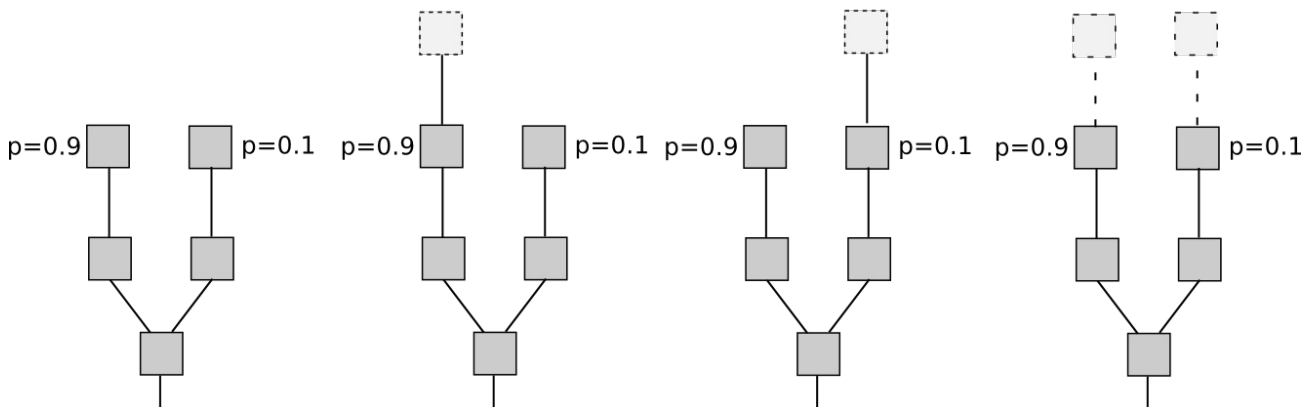
The intuition here is that we can replicate the economics of proof of work inside of proof of stake. In proof of work, there is also a penalty for creating a block on the wrong chain, but this penalty is implicit in the external environment: miners have to spend extra electricity and obtain or rent extra hardware. Here, we simply make the penalties explicit. This mechanism has the disadvantage that it imposes slightly more risk on validators (although the effect should be smoothed out over time), but has the advantage that it does not require validators to be known ahead of time.

That shows how chain-based algorithms solve nothing-at-stake. Now how do BFT-style proof of stake algorithms work?

BFT-style (partially synchronous) proof of stake algorithms allow validators to "vote" on blocks by sending one or more types of signed messages, and specify two kinds of rules:

- **Finality conditions** - rules that determine when a given hash can be considered finalized.
- **Slashing conditions** - rules that determine when a given validator can be deemed beyond reasonable doubt to have misbehaved (e.g. voting for multiple conflicting blocks at the same time). If a validator triggers one of these rules, their entire deposit gets deleted.

To illustrate the different forms that slashing conditions can take, we will give two examples of slashing conditions (hereinafter, "2/3 of all validators" is shorthand for "2/3 of all validators weighted by deposited coins", and likewise for other fractions and percentages). In these examples, "PREPARE" and "COMMIT" should be understood as simply referring to two types of messages that validators can send.

1. If `MESSAGES` contains messages of the form `["COMMIT", HASH1, view]` and `["COMMIT", HASH2, view]` for the same `view` but differing `HASH1` and `HASH2` signed by the same validator, then that validator is slashed.
2. If `MESSAGES` contains a message of the form `["COMMIT", HASH, view1]`, then UNLESS either view1 = -1 or there also exist messages of the form `["PREPARE", HASH, view1, view2]` for some specific `view2`, where view2 < view1, signed by 2/3 of all validators, then the validator that made the COMMIT is slashed.

There are two important desiderata for a suitable set of slashing conditions to have:

- **Accountable safety** - if conflicting `HASH1` and `HASH2` (i.e. `HASH1` and `HASH2` are different, and neither is a descendant of the other) are finalized, then at least 1/3 of all validators must have violated some slashing condition.
- **Plausible liveness** - unless at least 1/3 of all validators have violated some slashing condition, there exists a set of messages that 2/3 of validators can produce that finalize some value.

If we have a set of slashing conditions that satisfies both properties, then we can incentivize participants to send messages, and start benefiting from economic finality.

What is "economic finality" in general?

Economic finality is the idea that once a block is finalized, or more generally once enough messages of certain types have been signed, then the only way that at any point in the future the canonical history will contain a conflicting block is if a large number of people are willing to burn very large amounts of money. If a

node sees that this condition has been met for a given block, then they have a very economically strong assurance that that block will always be part of the canonical history that everyone agrees on.

There are two "flavors" of economic finality:

1. A block can be economically finalized if a sufficient number of validators have signed cryptoeconomic claims of the form "I agree to lose X in all histories where block B is not included". This gives clients assurance that either (i) B is part of the canonical chain, or (ii) validators lost a large amount of money in order to trick them into thinking that this is the case.
2. A block can be economically finalized if a sufficient number of validators have signed messages expressing support for block B, and there is a mathematical proof that *if some B' != B is also finalized under the same definition* then validators lose a large amount of money. If clients see this, and also validate the chain, and validity plus finality is a sufficient condition for precedence in the canonical fork choice rule, then they get an assurance that either (i) B is part of the canonical chain, or (ii) validators lost a large amount of money in making a conflicting chain that was also finalized.

The two approaches to finality inherit from the two solutions to the nothing at stake problem: finality by penalizing incorrectness, and finality by penalizing equivocation. The main benefit of the first approach is that it is more light-client friendly and is simpler to reason about, and the main benefits of the second approach are that (i) it's easier to see that honest validators will not be punished, and (ii) griefing factors are more favorable to honest validators.

Casper follows the second flavor, though it is possible that an on-chain mechanism will be added where validators can voluntarily opt-in to signing finality messages of the first flavor, thereby enabling much more efficient light clients.

So how does this relate to Byzantine fault tolerance theory?

Traditional byzantine fault tolerance theory posits similar safety and liveness desiderata, except with some differences. First of all, traditional byzantine fault tolerance theory simply requires that safety is achieved if 2/3 of validators are *honest*. This is a strictly easier model to work in; traditional fault tolerance tries to prove "if mechanism M has a safety failure, then at least 1/3 of nodes are faulty", whereas our model tries to prove "if mechanism M has a safety failure, then at least 1/3 of nodes are faulty, *and you know which ones, even if you were offline at the time the failure took place*". From a liveness perspective, our model is the easier one, as we do not demand a proof that the network *will* come to consensus, we just demand a proof that it does not get *stuck*.

Fortunately, we can show the additional accountability requirement is not a particularly difficult one; in fact, with the right "protocol armor", we can convert *any* traditional partially synchronous or asynchronous Byzantine fault-tolerant algorithm into an accountable algorithm. The proof of this basically boils down to the fact that faults can be exhaustively categorized into a few classes, and each one of these classes is either accountable (i.e. if you commit that type of fault you can get caught, so we can make a slashing condition for it) or indistinguishable from latency (note that even the fault of sending messages too early is indistinguishable from latency, as one can model it by speeding up everyone's clocks and assigning the messages that *weren't* sent too early a higher latency).

What is "weak subjectivity"?

It is important to note that the mechanism of using deposits to ensure there is "something at stake" does lead to one change in the security model. Suppose that deposits are locked for four months, and can later be withdrawn. Suppose that an attempted 51% attack happens that reverts 10 days worth of transactions. The blocks created by the attackers can simply be imported into the main chain as proof-of-malfeasance (or "dunkles") and the validators can be punished. However, suppose that such an attack happens after six months. Then, even though the blocks can certainly be re-imported, by that time the malfeasant validators will be able to withdraw their deposits on the main chain, and so they cannot be punished.

To solve this problem, we introduce a "revert limit" - a rule that nodes must simply refuse to revert further back in time than the deposit length (i.e. in our example, four months), and we additionally require nodes to log on at least once every deposit length to have a secure view of the chain. Note that this rule is different from every other consensus rule in the protocol, in that it means that nodes may come to different conclusions depending on when they saw certain messages. The time that a node saw a given message may be different between different nodes; hence we consider this rule "subjective" (alternatively, one well-versed in Byzantine fault tolerance theory may view it as a kind of synchrony assumption).

However, the "subjectivity" here is very weak: in order for a node to get on the "wrong" chain, they must receive the original message four months later than they otherwise would have. This is only possible in two cases:

1. When a node connects to the blockchain for the first time.
2. If a node has been offline for more than four months.

We can solve (1) by making it the user's responsibility to authenticate the latest state out of band. They can do this by asking their friends, block explorers, businesses that they interact with, etc. for a recent block hash in the chain that they see as the canonical one. In practice, such a block hash may well simply come as part of the software they use to verify the blockchain; an attacker that can corrupt the checkpoint in the software can arguably just as easily corrupt the software itself, and no amount of pure cryptoeconomic verification can solve that problem. (2) does genuinely add an additional security requirement for nodes, though note once again that the possibility of hard forks and security vulnerabilities, and the requirement to stay up to date to know about them and install any needed software updates, exists in proof of work too.

Note that all of this is a problem only in the very limited case where a majority of previous stakeholders from some point in time collude to attack the network and create an alternate chain; most of the time we expect there will only be one canonical chain to choose from.

Can we try to automate the social authentication to reduce the load on users?

One approach is to bake it into natural user workflow: a BIP 70-style payment request could include a recent block hash, and the user's client software would make sure that they are on the same chain as the vendor before approving a payment (or for that matter, any on-chain interaction). The other is to use Jeff Coleman's universal hash time. If UHT is used, then a successful attack chain would need to be generated secretly *at the same time* as the legitimate chain was being built, requiring a majority of validators to secretly collude for that long.

Can one economically penalize censorship in proof of stake?

Unlike reverts, censorship is much more difficult to prove. The blockchain itself cannot directly tell the difference between "user A tried to send transaction X but it was unfairly censored", "user A tried to send transaction X but it never got in because the transaction fee was insufficient" and "user A never tried to send transaction X at all". See also a note on data availability and erasure codes. However, there are a number of techniques that can be used to mitigate censorship issues.

The first is censorship resistance by halting problem. In the weaker version of this scheme, the protocol is designed to be Turing-complete in such a way that a validator cannot even tell whether or not a given transaction will lead to an undesired action without spending a large amount of processing power executing the transaction, and thus opening itself up to denial-of-service attacks. This is what prevented the DAO soft fork.

In the stronger version of the scheme, transactions can trigger guaranteed effects at some point in the near to mid-term future. Hence, a user could send multiple transactions which interact with each other and with predicted third-party information to lead to some future event, but the validators cannot possibly tell that this is going to happen until the transactions are already included (and economically finalized) and it is far too late to stop them; even if all future transactions are excluded, the event that validators wish to halt would still take place. Note that in this scheme, validators could still try to prevent **all** transactions, or perhaps all transactions that do not come packaged with some formal proof that they do not lead to anything undesired, but this would entail forbidding a very wide class of transactions to the point of essentially breaking the entire system, which would cause validators to lose value as the price of the cryptocurrency in which their deposits are denominated would drop.

The second, described by Adam Back here, is to require transactions to be timelock-encrypted. Hence, validators will include the transactions without knowing the contents, and only later could the contents automatically be revealed, by which point once again it would be far too late to un-include the transactions. If validators were sufficiently malicious, however, they could simply only agree to include transactions that come with a cryptographic proof (e.g. ZK-SNARK) of what the decrypted version is; this would force users to download new client software, but an adversary could conveniently provide such client software for easy download, and in a game-theoretic model users would have the incentive to play along.

Perhaps the best that can be said in a proof-of-stake context is that users could also install a software update that includes a hard fork that deletes the malicious validators and this is not that much harder than installing a software update to make their transactions "censorship-friendly". Hence, all in all this scheme is also moderately effective, though it does come at the cost of slowing interaction with the blockchain down (note that the scheme must be mandatory to be effective; otherwise malicious validators could much more easily simply filter encrypted transactions without filtering the quicker unencrypted transactions).

A third alternative is to include censorship detection in the fork choice rule. The idea is simple. Nodes watch the network for transactions, and if they see a transaction that has a sufficiently high fee for a sufficient amount of time, then they assign a lower "score" to blockchains that do not include this transaction. If all nodes follow this strategy, then eventually a minority chain would automatically coalesce that includes the transactions, and all honest online nodes would follow it. The main weakness of such a scheme is that offline nodes would still follow the majority branch, and if the censorship is temporary and they log back on after the censorship ends then they would end up on a different branch from online nodes. Hence, this scheme should be viewed more as a tool to facilitate automated emergency coordination on a hard fork than something that would play an active role in day-to-day fork choice.

How does validator selection work, and what is stake grinding?

In any chain-based proof of stake algorithm, there is a need for some mechanism which randomly selects which validator out of the currently active validator set can make the next block. For example, if the currently active validator set consists of Alice with 40 ether, Bob with 30 ether, Charlie with 20 ether and David with 10 ether, then you want there to be a 40% chance that Alice will be the next block creator, 30% chance that Bob will be, etc (in practice, you want to randomly select not just one validator, but rather an infinite sequence of validators, so that if Alice doesn't show up there is someone who can replace her after some time, but this doesn't change the fundamental problem). In non-chain-based algorithms randomness is also often needed for different reasons.

"Stake grinding" is a class of attack where a validator performs some computation or takes some other step to try to bias the randomness in their own favor. For example:

1. In Peercoin, a validator could "grind" through many combinations of parameters and find favorable parameters that would increase the probability of their coins generating a valid block.
2. In one now-defunct implementation, the randomness for block N+1 was dependent on the signature of block N. This allowed a validator to repeatedly produce new signatures until they found one that allowed them to get the next block, thereby seizing control of the system forever.
3. In NXT, the randomness for block N+1 is dependent on the validator that creates block N. This allows a validator to manipulate the randomness by simply skipping an opportunity to create a block. This carries an opportunity cost equal to the block reward, but sometimes the new random seed would give the validator an above-average number of blocks over the next few dozen blocks. See here and here for a more detailed analysis.

1. and (2) are easy to solve; the general approach is to require validators to deposit their coins well in advance, and not to use information that can be easily manipulated as source data for the randomness. There are several main strategies for solving problems like (3). The first is to use schemes based on secret sharing or deterministic threshold signatures and have validators collaboratively generate the random value. These schemes are robust against all manipulation unless a majority of validators collude (in some cases though, depending on the implementation, between 33-50% of validators can interfere in the operation, leading to the protocol having a 67% liveness assumption).

The second is to use cryptoeconomic schemes where validators commit to information (i.e. publish sha3(x)) well in advance, and then must publish x in the block; x is then added into the randomness pool. There are two theoretical attack vectors against this:

1. Manipulate x at commitment time. This is impractical because the randomness result would take many actors' values into account, and if even one of them is honest then the output will be a uniform distribution. A uniform distribution XORed together with arbitrarily many arbitrarily biased distributions still gives a uniform distribution.
2. Selectively avoid publishing blocks. However, this attack costs one block reward of opportunity cost, and because the scheme prevents anyone from seeing any future validators except for the next, it almost never provides more than one block reward worth of revenue. The only exception is the case where, if a validator skips, the next validator in line AND the first child of that validator will both be the same validator; if these situations are a grave concern then we can punish skipping further via an explicit skipping penalty.

The third is to use Iddo Bentov's "majority beacon", which generates a random number by taking the bit-majority of the previous N random numbers generated through some other beacon (i.e. the first bit of the result is 1 if the majority of the first bits in the source numbers is 1 and otherwise it's 0, the second bit of the result is 1 if the majority of the second bits in the source numbers is 1 and otherwise it's 0, etc). This gives a cost-of-exploitation of $\sim C * sqrt(N)$ where C is the cost of exploitation of the underlying beacons. Hence, all in all, many known solutions to stake grinding exist; the problem is more like differential cryptanalysis than the halting problem - an annoyance that proof of stake designers eventually understood and now know how to overcome, not a fundamental and inescapable flaw.

What would the equivalent of a 51% attack against Casper look like?

There are four basic types of 51% attack:

- **Finality reversion**: validators that already finalized block A then finalize some competing block A', thereby breaking the blockchain's finality guarantee.
- **Invalid chain finalization**: validators finalize an invalid (or unavailable) block.
- **Liveness denial**: validators stop finalizing blocks.
- **Censorship**: validators block some or all transactions or blocks from entering the chain.

In the first case, users can socially coordinate out-of-band to agree which finalized block came first, and favor that block. The second case can be solved with fraud proofs and data availability proofs. The third case can be solved by a modification to proof of stake algorithms that gradually reduces ("leaks") non-participating nodes' weights in the validator set if they do not participate in consensus; the Casper FFG paper includes a description of this.

The fourth is most difficult. The fourth can be recovered from via a "minority soft fork", where a minority of honest validators agree the majority is censoring them, and stop building on their chain. Instead, they continue their own chain, and eventually the "leak" mechanism described above ensures that this honest minority becomes a 2/3 supermajority on the new chain. At that point, the market is expected to favor the chain controlled by honest nodes over the chain controlled by dishonest nodes.

That sounds like a lot of reliance on out-of-band social coordination; is that not dangerous?

Attacks against Casper are extremely expensive; as we will see below, attacks against Casper cost as much, if not more, than the cost of buying enough mining power in a proof of work chain to permanently 51% attack it over and over again to the point of uselessness. Hence, the recovery techniques described above will only be used in very extreme circumstances; in fact, advocates of proof of work also generally express willingness to use social coordination in similar circumstances by, for example, changing the proof of work algorithm. Hence, it is not even clear that the need for social coordination in proof of stake is larger than it is in proof of work.

In reality, we expect the amount of social coordination required to be near-zero, as attackers will realize that it is not in their benefit to burn such large amounts of money to simply take a blockchain offline for one or two days.

Doesn't MC <= MR mean that all consensus algorithms with a given security level are equally efficient (or in other words, equally wasteful)?

This is an argument that many have raised, perhaps best explained by Paul Sztorc in this article. Essentially, if you create a way for people to earn $100, then people will be willing to spend anywhere up to $99.9 (including the cost of their own labor) in order to get it; marginal cost approaches marginal revenue. Hence, the theory goes, any algorithm with a given block reward will be equally "wasteful" in terms of the quantity of socially unproductive activity that is carried out in order to try to get the reward.

There are three flaws with this:

1. It's not enough to simply say that marginal cost approaches marginal revenue; one must also posit a plausible mechanism by which someone can actually expend that cost. For example, if tomorrow I announce that every day from then on I will give $100 to a randomly selected one of a given list of ten people (using my laptop's /dev/urandom as randomness), then there is simply no way for anyone to send $99 to try to get at that randomness. Either they are not in the list of ten, in which case they have no chance no matter what they do, or they are in the list of ten, in which case they don't have any reasonable way to manipulate my randomness so they're stuck with getting the expected-value $10 per day.
2. MC <= MR does NOT imply total cost approaches total revenue. For example, suppose that there is an algorithm which pseudorandomly selects 1000 validators out of some very large set (each validator getting a reward of $1), you have 10% of the stake so on average you get 100, and at a cost of $1 you can force the randomness to reset (and you can repeat this an unlimited number of times). Due to the central limit theorem, the standard deviation of your reward is $10, and based on other known results in math the expected maximum of N random samples is slightly under $M + S * sqrt(2 * log(N))$ where M is the mean and S is the standard deviation. Hence the reward for making additional trials (i.e. increasing N) drops off sharply, e.g. with 0 re-trials your expected reward is $100, with one re-trial it's $105.5, with two it's $108.5, with three it's $110.3, with four it's $111.6, with five it's $112.6 and with six it's $113.5. Hence, after five retrials it stops being worth it. As a result, an economically motivated attacker with ten percent of stake will inefficiently spend $5 to get an additional revenue of $13, though the total revenue is $113. If the exploitable mechanisms only expose small opportunities, the economic loss will be small; it is decidedly NOT the case that a single drop of exploitability brings the entire flood of PoW-level economic waste rushing back in. This point will also be very relevant in our below discussion on capital lockup costs.
3. Proof of stake can be secured with much lower total rewards than proof of work.

What about capital lockup costs?

Locking up X ether in a deposit is not free; it entails a sacrifice of optionality for the ether holder. Right now, if I have 1000 ether, I can do whatever I want with it; if I lock it up in a deposit, then it's stuck there for months, and I do not have, for example, the insurance utility of the money being there to pay for sudden unexpected expenses. I also lose some freedom to change my token allocations away from ether within that timeframe; I could simulate selling ether by shorting an amount equivalent to the deposit on an exchange, but this itself carries costs including exchange fees and paying interest. Some might argue: isn't this capital lockup inefficiency really just a highly indirect way of achieving the exact same level of economic inefficiency as exists in proof of work? The answer is no, for

both reasons (2) and (3) above.

Let us start with (3) first. Consider a model where proof of stake deposits are infinite-term, ASICs last forever, ASIC technology is fixed (i.e. no Moore's law) and electricity costs are zero. Let's say the equilibrium interest rate is 5% per annum. In a proof of work blockchain, I can take $1000, convert it into a miner, and the miner will pay me $50 in rewards per year forever. In a proof of stake blockchain, I would buy $1000 of coins, deposit them (i.e. losing them forever), and get $50 in rewards per year forever. So far, the situation looks completely symmetrical (technically, even here, in the proof of stake case my destruction of coins isn't fully socially destructive as it makes others' coins worth more, but we can leave that aside for the moment). The cost of a "Maginot-line" 51% attack (i.e. buying up more hardware than the rest of the network) increases by $1000 in both cases.

Now, let's perform the following changes to our model in turn:

1. Moore's law exists, ASICs depreciate by 50% every 2.772 years (that's a continuously-compounded 25% annual depreciation; picked to make the numbers simpler). If I want to retain the same "pay once, get money forever" behavior, I can do so: I would put $1000 into a fund, where $167 would go into an ASIC and the remaining $833 would go into investments at 5% interest; the $41.67 dividends per year would be just enough to keep renewing the ASIC hardware (assuming technological development is fully continuous, once again to make the math simpler). Rewards would go down to $8.33 per year; hence, 83.3% of miners will drop out until the system comes back into equilibrium with me earning $50 per year, and so the Maginot-line cost of an attack on PoW given the same rewards drops by a factor of 6.

2. Electricity plus maintenance makes up 1/3 of mining costs. We estimate the 1/3 from recent mining statistics: one of Bitfury's new data centers consumes 0.06 joules per gigahash, or 60 J/TH or 0.000017 kWh/TH, and if we assume the entire Bitcoin network has similar efficiencies we get 27.9 kWh per second given 1.67 million TH/s total Bitcoin hashpower. Electricity in China costs $0.11 per kWh, so that's about $3 per second, or $260,000 per day. Bitcoin block rewards plus fees are $600 per BTC * 13 BTC per block * 144 blocks per day = $1.12m per day. Thus electricity itself would make up 23% of costs, and we can back-of-the-envelope estimate maintenance at 10% to give a clean 1/3 ongoing costs, 2/3 fixed costs split. This means that out of your $1000 fund, only $111 would go into the ASIC, $56 would go into paying ongoing costs, and $833 would go into investments; hence the Maginot-line cost of attack is 9x lower than in our original setting.

3. Deposits are temporary, not permanent. Sure, if I voluntarily keep staking forever, then this changes nothing. However, I regain some of the optionality that I had before; I could quit within a medium timeframe (say, 4 months) at any time. This means that I would be willing to put more than $1000 of ether in for the $50 per year gain; perhaps in equilibrium it would be something like $3000. Hence, the cost of the Maginot line attack on PoS *increases* by a factor of three, and so on net PoS gives 27x more security than PoW for the same cost.

The above included a large amount of simplified modeling, however it serves to show how multiple factors stack up heavily in favor of PoS in such a way that PoS gets *more* bang for its buck in terms of security. The meta-argument for why this perhaps suspiciously multifactorial argument leans so heavily in favor of PoS is simple: in PoW, we are working directly with the laws of physics. In PoS, we are able to design the protocol in such a way that it has the precise properties that we want - in short, we can *optimize the laws of physics in our favor*. The "hidden trapdoor" that gives us (3) is the change in the security model, specifically the introduction of weak subjectivity.

Now, we can talk about the marginal/total distinction. In the case of capital lockup costs, this is very important. For example, consider a case where you have $100,000 of ether. You probably intend to hold a large portion of it for a long time; hence, locking up even $50,000 of the ether should be nearly free. Locking up $80,000 would be slightly more inconvenient, but $20,000 of breathing room still gives you a large space to maneuver. Locking up $90,000 is more problematic, $99,000 is very problematic, and locking up all $100,000 is absurd, as it means you would not even have a single bit of ether left to pay basic transaction fees. Hence, your marginal costs increase quickly. We can show the difference between this state of affairs and the state of affairs in proof of work as follows:

Hence, the *total* cost of proof of stake is potentially much lower than the marginal cost of depositing 1 more ETH into the system multiplied by the amount of ether currently deposited.

Note that this component of the argument unfortunately does not fully translate into reduction of the "safe level of issuance". It does help us because it shows that we can get substantial proof of stake participation even if we keep issuance very low; however, it also means that a large portion of the gains will simply be borne by validators as economic surplus.

Will exchanges in proof of stake pose a similar centralization risk to pools in proof of work?

From a centralization perspective, in both Bitcoin and Ethereum it's the case that roughly three pools are needed to coordinate on a 51% attack (4 in Bitcoin, 3 in Ethereum at the time of this writing). In PoS, if we assume 30% participation including all exchanges, then three exchanges would be enough to make a 51% attack; if participation goes up to 40% then the required number goes up to eight. However, exchanges will not be able to participate with all of their ether; the reason is that they need to accomodate withdrawals.

Additionally, pooling in PoS is discouraged because it has a much higher trust requirement - a proof of stake pool can pretend to be hacked, destroy its participants' deposits and claim a reward for it. On the other hand, the ability to earn interest on one's coins without oneself running a node, even if trust is required, is something that many may find attractive; all in all, the centralization balance is an empirical question for which the answer is unclear until the system is actually running for a substantial period of time. With sharding, we expect pooling incentives to reduce further, as (i) there is even less concern about variance, and (ii) in a sharded model, transaction verification load is proportional to the amount of capital that one puts in, and so there are no direct infrastructure savings from pooling.

A final point is that centralization is less harmful in proof of stake than in proof of work, as there are much cheaper ways to recover from successful 51% attacks; one does not need to switch to a new mining algorithm.

Are there economic ways to discourage centralization?

One strategy suggested by Vlad Zamfir is to only partially destroy deposits of validators that get slashed, setting the percentage destroyed to be proportional to the percentage of other validators that have been slashed recently. This ensures that validators lose all of their deposits in the event of an actual attack, but only a small part of their deposits in the event of a one-off mistake. This makes lower-security staking strategies possible, and also specifically incentivizes validators to have their errors be as uncorrelated (or ideally, anti-correlated) with other validators as possible; this involves not being in the largest pool, putting one's node on the largest virtual private server provider and even using secondary software implementations, all of which increase decentralization.

Can proof of stake be used in private/consortium chains?

Generally, yes; any proof of stake algorithm can be used as a consensus algorithm in private/consortium chain settings. The only change is that the way the validator set is selected would be different: it would start off as a set of trusted users that everyone agrees on, and then it would be up to the validator set to vote on adding in new validators.

Can multi-currency proof of stake work?

There has been a lot of interest in proof of stake protocols where users can stake any currency, or one of multiple currencies. However, these designs unfortunately introduce economic challenges that likely make them much more trouble than any benefit that could be received from them. The key problems include:

- **Price oracle dependence**: if people are staking in multiple cryptocurrencies, there needs to be a way to compare deposits in one versus the other, so as to fairly allocate proposal rights, determine whether or not a 2/3 threshold was passed, etc. This requires some form of price oracle. This can be done in a decentralized way (eg. see Uniswap), but it introduces another component that could be manipulated and attacked by validators.
- **Pathological cryptocurrencies**: one can always create a cryptocurrency that is pathologically constructed to nullify the impact of penalties. For example, one can imagine a fiat-backed token where coins that are seized by the protocol as penalties are tracked and not honored by the issuer, and the penalized actor's original balance is honored instead. This logic could even be implemented in a smart contract, and it's impossible to determine with certainty whether or not a given currency has such a mechanism built-in.
- **Reduced incentive alignment**: if currencies other than the protocol's base token can be used to stake, this reduces the stakers' interest in seeing the protocol continue to operate and succeed.

Further reading

https://github.com/ethereum/wiki/wiki/Casper-Proof-of-Stake-compendium

Sharding FAQ

Currently, in all blockchain protocols each node stores the entire state (account balances, contract code and storage, etc.) and processes all transactions. This provides a large amount of security, but greatly limits scalability: a blockchain cannot process more transactions than a single node can. In large part because of this, Bitcoin is limited to ~3–7 transactions per second, Ethereum to 7–15, etc.

However, this poses a question: are there ways to create a new mechanism, where only a small subset of nodes verifies each transaction? As long as there are sufficiently many nodes verifying each transaction that the system is still highly secure, but a sufficiently small percentage of the total validator set so that the system can process many transactions in parallel, could we not split up transaction processing between smaller groups of nodes to greatly increase a blockchain's total throughput?

Contents

What are some trivial but flawed ways of solving the problem?

There are three main categories of "easy solutions". The first is to give up on scaling individual blockchains, and instead assume that applications will be split among many different chains. This greatly increases throughput, but at a cost of security: an N-factor increase in throughput using this method necessarily comes with an N-factor decrease in security, as a level of resources 1/N the size of the whole ecosystem will be sufficient to attack any individual chain. Hence, it is arguably non-viable for more than small values of N.

The second is to simply increase the block size limit. This can work and in some situations may well be the correct prescription, as block sizes may well be constrained more by politics than by realistic technical considerations. But regardless of one's beliefs about any individual case such an approach inevitably has its limits: if one goes too far, then nodes running on consumer hardware will drop out, the network will start to rely exclusively on a very small number of supercomputers running the blockchain, which can lead to great centralization risk.

The third is "merge mining", a technique where there are many chains, but all chains share the same mining power (or, in proof of stake systems, stake). Currently, Namecoin gets a large portion of its security from the Bitcoin blockchain by doing this. If all miners participate, this theoretically can increase throughput by a factor of N without compromising security. However, this also has the problem that it increases the computational and storage load on each miner by a factor of N, and so in fact this solution is simply a stealthy form of block size increase.

This sounds like there's some kind of scalability trilemma at play. What is this trilemma and can we break through it?

The trilemma claims that blockchain systems can only at most have two of the following three properties:

- **Decentralization** (defined as the system being able to run in a scenario where each participant only has access to $O(c)$ resources, i.e. a regular laptop or small VPS)
- **Scalability** (defined as being able to process $O(n) > O(c)$ transactions)
- **Security** (defined as being secure against attackers with up to $O(n)$ resources)

In the rest of this document, we'll continue using **c** to refer to the size of computational resources (including computation, bandwidth and storage) available to each node, and **n** to refer to the size of the ecosystem in some abstract sense; we assume that transaction load, state size, and the market cap of a cryptocurrency are all proportional to **n**. The key challenge of scalability is finding a way to achieve all three at the base layer.

What are some moderately simple but only partial ways of solving the scalability problem?

Many sharding proposals (e.g. this early BFT sharding proposal from Loi Luu et al at NUS, more recent application of similar ideas in Zilliqa, as well as this Merklix tree[1] approach that has been

suggested for Bitcoin) attempt to either only shard transaction processing or only shard state, without touching the other[2]. These efforts can lead to some gains in efficiency, but they run into the fundamental problem that they only solve one of the two bottlenecks. We want to be able to process 10,000+ transactions per second without either forcing every node to be a supercomputer or forcing every node to store a terabyte of state data, and this requires a comprehensive solution where the workloads of state storage, transaction processing and even transaction downloading and re-broadcasting at are all spread out across nodes. Particularly, the P2P network needs to also be modified to ensure that not every node receives all information from every other node.

What about approaches that do not try to "shard" anything?

Bitcoin-NG can increase scalability somewhat by means of an alternative blockchain design that makes it much safer for the network if nodes are spending large portions of their CPU time verifying blocks. In simple PoW blockchains, there are high centralization risks and the safety of consensus is weakened if capacity is increased to the point where more than about 5% of nodes' CPU time is spent verifying blocks; Bitcoin-NG's design alleviates this problem. However, this can only increase the scalability of transaction capacity by a constant factor of perhaps 5-50x[3,4], and does not increase the scalability of state. That said, Bitcoin-NG-style approaches are not mutually exclusive with sharding, and the two can certainly be implemented at the same time.

Channel-based strategies (lightning network, Raiden, etc) can scale transaction capacity by a constant factor but cannot scale state storage, and also come with their own unique sets of tradeoffs and limitations particularly involving denial-of-service attacks. On-chain scaling via sharding (plus other techniques) and off-chain scaling via channels are arguably both necessary and complementary.

There exist approaches that use advanced cryptography, such as Mimblewimble and strategies based on ZK-SNARKs (eg. Coda), to solve one specific part of the scaling problem: initial full node synchronization. Instead of verifying the entire history from genesis, nodes could verify a cryptographic proof that the current state legitimately follows from the history. These approaches do solve a legitimate problem, but they are not a substitute for sharding, as they do not remove the need for nodes to download and verify very large amounts of data to stay on the chain in real time.

How does Plasma, state channels and other layer 2 technologies fit into the trilemma?

In the event of a large attack on Plasma subchains, all users of the Plasma subchains would need to withdraw back to the root chain. If Plasma has O(N) users, then this will require O(N) transactions, and so O(N / C) time to process all of the withdrawals. If withdrawal delays are fixed to some D (i.e. the naive implementation), then as soon as N > C * D, there will not be enough space in the blockchain to process all withdrawals in time, and so the system will be insecure; in this mode, Plasma should be viewed as increasing scalability only by a (possibly large) constant factor. If withdrawal delays are flexible, so they automatically extend if there are many withdrawals being made, then this means that as N increases further and further, the amount of time that an attacker can force everyone's funds to get locked up increases, and so the level of "security" of the system decreases further and further in a certain sense, as extended denial of access can be viewed as a security failure, albeit one milder than total loss of access. However, this is a different *direction* of tradeoff from other solutions, and arguably a much milder tradeoff, hence why Plasma subchains are nevertheless a large improvement on the status quo.

Note that there is one design that states that: "Given a malicious operator (the worst case), the system degrades to an on-chain token. A malicious operator cannot steal funds and cannot deprive people of their funds for any meaningful amount of time."—https://ethresear.ch/t/roll-up-roll-back-snark-side-chain-17000-tps/3675. See also here for related information.

State channels have similar properties, though with different tradeoffs between versatility and speed of finality. Other layer 2 technologies include TrueBit off-chain interactive verification of execution and Raiden, which is another organisation working on state channels. Proof of stake with Casper (which is layer 1) would also improve scaling—it is more decentralizable, not requiring a computer that is able to mine, which tends towards centralized mining farms and institutionalized mining pools as difficulty increases and the size of the state of the blockchain increases.

Sharding is different to state channels and Plasma in that periodically notaries are pseudo-randomly assigned to vote on the validity of collations (analogous to blocks, but without an EVM state transition function in phase 1), then these collations are accepted into the main chain after the votes are verified by a committee on the main chain, via a sharding manager contract on the main chain. In phase 5 (see the [roadmap](#) for details), shards are tightly coupled to the main chain, so that if any shard or the main chain is invalid, the whole network is invalid. There are other differences between each mechanism, but at a high level, Plasma, state channels and Truebit are off-chain for an indefinite interval, connect to the main chain at the smart contract, layer 2 level, while they can draw back into and open up from the main chain, whereas shards are regularly linked to the main chain via consensus in-protocol.

See also [these tweets from Vlad](#).

State size, history, cryptoeconomics, oh my! Define some of these terms before we move further!

- **State**: a set of information that represents the "current state" of a system; determining whether or not a transaction is valid, as well as the effect of a transaction, should in the simplest model depend only on state. Examples of state data include the UTXO set in bitcoin, balances + nonces + code + storage in ethereum, and domain name registry entries in Namecoin.
- **History**: an ordered list of all transactions that have taken place since genesis. In a simple model, the present state should be a deterministic function of the genesis state and the history.
- **Transaction**: an object that goes into the history. In practice, a transaction represents an operation that some user wants to make, and is cryptographically signed. In some systems transactions are called **blobs**, to emphasize the fact that in these systems these objects may contain arbitrary data and may not in all cases represent an attempt to perform some operation in the protocol.
- **State transition function**: a function that takes a state, applies a transaction and outputs a new state. The computation involved may involve adding and subtracting balances from accounts specified by the transaction, verifying digital signatures and running contract code.
- **Merkle tree**: a cryptographic hash tree structure that can store a very large amount of data, where authenticating each individual piece of data only takes $O(\log(n))$ space and time. See [here](#) for details. In Ethereum, the transaction set of each block, as well as the state, is kept in a Merkle tree, where the roots of the trees are committed to in a block.
- **Receipt**: an object that represents an effect of a transaction that is not directly stored in the state, but which is still stored in a Merkle tree and committed to in a block header or in a special location in the state so that its existence can later be efficiently proven even to a node that does not have all of the data. Logs in Ethereum are receipts; in sharded models, receipts are used to facilitate asynchronous cross-shard communication.
- **Light client**: a way of interacting with a blockchain that only requires a very small amount (we'll say $O(1)$, though $O(\log(c))$ may also be accurate in some cases) of computational resources, keeping track of only the block headers of the chain by default and acquiring any needed information about transactions, state or receipts by asking for and verifying Merkle proofs of the relevant data on an as-needed basis.
- **State root**: the root hash of the Merkle tree representing the state[5]
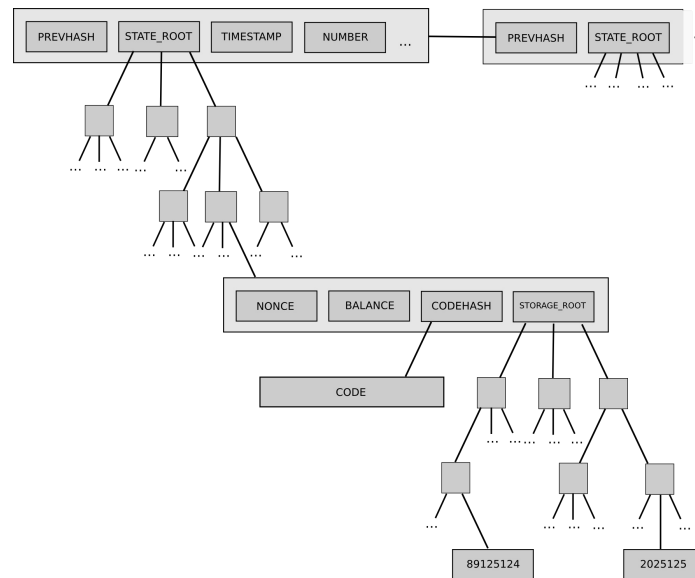
The Ethereum 1.0 state tree, and how the state root fits into the block structure

What is the basic idea behind sharding?

We split the state and history up into K = O(n / c) partitions that we call "shards". For example, a sharding scheme on Ethereum might put all addresses starting with 0x00 into one shard, all addresses starting with 0x01 into another shard, etc. In the simplest form of sharding, each shard also has its own transaction history, and the effect of transactions in some shard k are limited to the state of shard k. One simple example would be a multi-asset blockchain, where there are K shards and each shard stores the balances and processes the transactions associated with one particular asset. In more advanced forms of sharding, some form of cross-shard communication capability, where transactions on one shard can trigger events on other shards, is also included.

What might a basic design of a sharded blockchain look like?

A simple approach is as follows. For simplicity, this design keeps track of data blobs only; it does not attempt to process a state transition function.

There exists a set of **validators** (ie. proof of stake nodes), who randomly get assigned the right to create **shard blocks**. During each **slot** (eg. an 8-second period of time), for each k in [0...999] a random validator gets selected, and given the right to create a block on "shard k", which might contain up to, say, 32 kb of data. Also, for each k, a set of 100 validators get selected as **attesters**. The header of a block together with at least 67 of the attesting signatures can be published as an object that gets included in the "main chain" (also called a **beacon chain**).

Note that there are now several "levels" of nodes that can exist in such a system:

- **Super-full node** - downloads the full data of the beacon chain and every shard block referenced in the beacon chain.
- **Top-level node** - processes the beacon chain blocks only, including the headers and signatures of the shard blocks, but does not download all the data of the shard blocks.
- **Single-shard node** - acts as a top-level node, but also fully downloads and verifies every collation on some specific shard that it cares more about.
- **Light node** - downloads and verifies the block headers of main chain blocks only; does not process any collation headers or transactions unless it needs to read some specific entry in the state of some specific shard, in which case it downloads the Merkle branch to the most recent collation header for that shard and from there downloads the Merkle proof of the desired value in the state.

What are the challenges here?

- **Single-shard takeover attacks** - what if an attacker takes over the majority of the validators responsible for attesting to one particular block, either to (respectively) prevent any collations from getting enough signatures or, worse, to submit collations that are invalid?
- **State transition execution** - single-shard takeover attacks are typically prevented with random sampling schemes, but such schemes also make it more difficult for validators to compute state roots, as they cannot have up-to-date state information for every shard that they could be assigned to. How do we ensure that light clients can still get accurate information about the state?
- **Fraud detection** - if an invalid collation or state claim does get made, how can nodes (including light nodes) be reliably informed of this so that they can detect the fraud and reject the collation if it is truly fraudulent?
- **Cross shard communication** - the above design supports no cross-shard communication. How do we add cross-shard communication safely?
- **The data availability problem** - as a subset of fraud detection, what about the specific case where data is missing from a collation?
- **Superquadratic sharding** - in the special case where $n > c^2$, in the simple design given above there would be more than $O(c)$ collation headers, and so an ordinary node would not be able to process even just the top-level blocks. Hence, more than two levels of indirection between transactions and top-level block headers are required (i.e. we need "shards of shards"). What is the simplest and best way to do this?

However, the effect of a transaction may depend on *events that earlier took place in other shards*; a canonical example is transfer of money, where money can be moved from shard i to shard j by first creating a "debit" transaction that destroys coins in shard i, and then creating a "credit" transaction that creates coins in shard j, pointing to a receipt created by the debit transaction as proof that the credit is legitimate.

But doesn't the CAP theorem mean that fully secure distributed systems are impossible, and so sharding is futile?

The CAP theorem is a result that has to do with *distributed consensus*; a simple statement is: "in the cases that a network partition takes place, you have to choose either consistency or availability, you cannot have both". The intuitive argument is simple: if the network splits in half, and in one half I send a transaction "send my 10 coins to A" and in the other I send a transaction "send my 10 coins to B", then either the system is unavailable, as one or both transactions will not be processed, or it becomes inconsistent, as one half of the network will see the first transaction completed and the other half will see the second transaction completed. Note that the CAP theorem has nothing to do with scalability; it applies to any situation where multiple nodes need to agree on a value, regardless of the amount of data that they are agreeing on. All existing decentralized systems have found some compromise between availability and consistency; sharding does not make anything fundamentally harder in this respect.

What are the security models that we are operating under?

There are several competing models under which the safety of blockchain designs is evaluated:

- **Honest majority** (or honest supermajority): we assume that there is some set of validators and up to 50% (or 33% or 25%) of those validators are controlled by an attacker, and the remaining validators honestly follow the protocol. Honest majority models can have **non-adaptive** or **adaptive** adversaries; an adversary is adaptive if they can quickly choose which portion of the validator set to "corrupt", and non-adaptive if they can only make that choice far ahead of time. Note that the honest majority assumption may be higher for notary committees with a [61% honesty assumption](#).
- **Uncoordinated majority**: we assume that all validators are rational in a game-theoretic sense (except the attacker, who is motivated to make the network fail in some way), but no more than some fraction (often between 25% and 50%) are capable of coordinating their actions.
- **Coordinated choice**: we assume that most or all validators are controlled by the same actor, or are fully capable of coordinating on the economically optimal choice between themselves. We

can talk about the **cost to the coalition** (or profit to the coalition) of achieving some undesirable outcome.

- **Bribing attacker model**: we take the uncoordinated majority model, but instead of making the attacker be one of the participants, the attacker sits outside the protocol, and has the ability to bribe any participants to change their behavior. Attackers are modeled as having a **budget**, which is the maximum that they are willing to pay, and we can talk about their **cost**, the amount that they *end up paying* to disrupt the protocol equilibrium.

Bitcoin proof of work with [Eyal and Sirer's selfish mining fix](#) is robust up to 50% under the honest majority assumption, and up to ~23.21% under the uncoordinated majority assumption. [Schellingcoin](#) is robust up to 50% under the honest majority and uncoordinated majority assumptions, has ε (i.e. slightly more than zero) cost of attack in a coordinated choice model, and has a P + ε budget requirement and ε cost in a bribing attacker model due to [P + epsilon attacks](#).

Hybrid models also exist; for example, even in the coordinated choice and bribing attacker models, it is common to make an **honest minority assumption** that some portion (perhaps 1-15%) of validators will act altruistically regardless of incentives. We can also talk about coalitions consisting of between 50-99% of validators either trying to disrupt the protocol or harm other validators; for example, in proof of work, a 51%-sized coalition can double its revenue by refusing to include blocks from all other miners.

The honest majority model is arguably highly unrealistic and has already been empirically disproven - see Bitcoin's [SPV mining fork](#) for a practical example. It proves too much: for example, an honest majority model would imply that honest miners are willing to voluntarily burn their own money if doing so punishes attackers in some way. The uncoordinated majority assumption may be realistic; there is also an intermediate model where the majority of nodes is honest but has a budget, so they shut down if they start to lose too much money.

The bribing attacker model has in some cases been criticized as being unrealistically adversarial, although its proponents argue that if a protocol is designed with the bribing attacker model in mind then it should be able to massively reduce the cost of consensus, as 51% attacks become an event that could be recovered from. We will evaluate sharding in the context of both uncoordinated majority and bribing attacker models. Bribing attacker models are similar to maximally-adaptive adversary models, except that the adversary has the additional power that it can solicit private information from all nodes; this distinction can be crucial, for example [Algorand](#) is secure under adaptive adversary models but not bribing attacker models because of how it relies on private information for random selection.

How can we solve the single-shard takeover attack in an uncoordinated majority model?

In short, random sampling. Each shard is assigned a certain number of notaries (e.g. 150), and the notaries that approve collations on each shard are taken from the sample for that shard. Samples can be reshuffled either semi-frequently (e.g. once every 12 hours) or maximally frequently (i.e. there is no real independent sampling process, notaries are randomly selected for each shard from a global pool every block).

Sampling can be explicit, as in protocols that choose specifically sized "committees" and ask them to vote on the validity and availability of specific collations, or it can be implicit, as in the case of "longest chain" protocols where nodes pseudorandomly assigned to build on specific collations and are expected to "windback verify" at least N ancestors of the collation they are building on.

The result is that even though only a few nodes are verifying and creating blocks on each shard at any given time, the level of security is in fact not much lower, in an honest or uncoordinated majority model, than what it would be if every single node was verifying and creating blocks. The reason is simple statistics: if you assume a ~67% honest supermajority on the global set, and if the size of the sample is 150, then with 99.999% probability the honest majority condition will be satisfied on the sample. If you assume a 75% honest supermajority on the global set, then that probability increases to 99.999999998% (see [here](#) for calculation details).

Hence, at least in the honest / uncoordinated majority setting, we have:

- **Decentralization** (each node stores only O(c) data, as it's a light client in O(c) shards and so stores O(1) * O(c) = O(c) data worth of block headers, as well as O(c) data corresponding to the recent history of one or several shards that it is assigned to at the present time)

- **Scalability** (with O(c) shards, each shard having O(c) capacity, the maximum capacity is n = O(c^2))
- **Security** (attackers need to control at least ~33% of the entire O(n)-sized validator pool in order to stand a chance of taking over the network).

In the bribing attacker model (or in the "very very adaptive adversary" model), things are not so easy, but we will get to this later. Note that because of the imperfections of sampling, the security threshold does decrease from 50% to ~30-40%, but this is still a surprisingly low loss of security for what may be a 100-1000x gain in scalability with no loss of decentralization.

How do you actually do this sampling in proof of work, and in proof of stake?

In proof of stake, it is easy. There already is an "active validator set" that is kept track of in the state, and one can simply sample from this set directly. Either an in-protocol algorithm runs and chooses 150 validators for each shard, or each validator independently runs an algorithm that uses a common source of randomness to (provably) determine which shard they are at any given time. Note that it is very important that the sampling assignment is "compulsory"; validators do not have a choice of what shard they go into. If validators could choose, then attackers with small total stake could concentrate their stake onto one shard and attack it, thereby eliminating the system's security.

In proof of work, it is more difficult, as with "direct" proof of work schemes one cannot prevent miners from applying their work to a given shard. It may be possible to use [proof-of-file-access forms](#) of proof of work to lock individual miners to individual shards, but it is hard to ensure that miners cannot quickly download or generate data that can be used for other shards and thus circumvent such a mechanism. The best known approach is through a technique invented by Dominic Williams called "puzzle towers", where miners first perform proof of work on a common chain, which then inducts them into a proof of stake-style validator pool, and the validator pool is then sampled just as in the proof-of-stake case.

One possible intermediate route might look as follows. Miners can spend a large (O(c)-sized) amount of work to create a new "cryptographic identity". The precise value of the proof of work solution then chooses which shard they have to make their next block on. They can then spend an O(1)-sized amount of work to create a block on that shard, and the value of that proof of work solution determines which shard they can work on next, and so on[8]. Note that all of these approaches make proof of work "stateful" in some way; the necessity of this is fundamental.

How is the randomness for random sampling generated?

First of all, it is important to note that even if random number generation is heavily exploitable, this is not a fatal flaw for the protocol; rather, it simply means that there is a medium to high centralization incentive. The reason is that because the randomness is picking fairly large samples, it is difficult to bias the randomness by more than a certain amount.

The simplest way to show this is through the [binomial distribution](#), as described above; if one wishes to avoid a sample of size N being more than 50% corrupted by an attacker, and an attacker has p% of the global stake pool, the chance of the attacker being able to get such a majority during one round is:

$$\sum_{k=\frac{N}{2}}^{N} p^k (1-p)^{N-k} \binom{N}{k}$$

Here's a table for what this probability would look like in practice for various values of N and p:

<

	N = 50	N = 100	N = 150	N = 250
p = 0.4	0.0978	0.0271	0.0082	0.0009
p = 0.33	0.0108	0.0004	$1.83 * 10^{-5}$	$3.98 * 10^{-8}$
p = 0.25	0.0001	$6.63 * 10^{-8}$	$4.11 * 10^{-11}$	$1.81 * 10^{-17}$

p = 0.2 $2.09 * 10^{-6}$ $2.14 * 10^{-11}$ $2.50 * 10^{-16}$ $3.96 * 10^{-26}$

Hence, for N >= 150, the chance that any given random seed will lead to a sample favoring the attacker is very small indeed[11,12]. What this means from the perspective of security of randomness is that the attacker needs to have a very large amount of freedom in choosing the random values order to break the sampling process outright. Most vulnerabilities in proof-of-stake randomness do not allow the attacker to simply choose a seed; at worst, they give the attacker many chances to select the most favorable seed out of many pseudorandomly generated options. If one is very worried about this, one can simply set N to a greater value, and add a moderately hard key-derivation function to the process of computing the randomness, so that it takes more than 2^{100} computational steps to find a way to bias the randomness sufficiently.

Now, let's look at the risk of attacks being made that try to influence the randomness more marginally, for purposes of profit rather than outright takeover. For example, suppose that there is an algorithm which pseudorandomly selects 1000 validators out of some very large set (each validator getting a reward of $1), an attacker has 10% of the stake so the attacker's average "honest" revenue 100, and at a cost of $1 the attacker can manipulate the randomness to "re-roll the dice" (and the attacker can do this an unlimited number of times).

Due to the [central limit theorem](), the standard deviation of the number of samples, and based [on other known results in math]() the expected maximum of N random samples is slightly under M + S * sqrt(2 * log(N)) where M is the mean and S is the standard deviation. Hence the reward for manipulating the randomness and effectively re-rolling the dice (i.e. increasing N) drops off sharply, e.g. with 0 re-trials your expected reward is $100, with one re-trial it's $105.5, with two it's $108.5, with three it's $110.3, with four it's $111.6, with five it's $112.6 and with six it's $113.5. Hence, after five retrials it stops being worth it. As a result, an economically motivated attacker with ten percent of stake will (socially wastefully) spend $5 to get an additional revenue of $13, for a net surplus of $8.

However, this kind of logic assumes that one single round of re-rolling the dice is expensive. Many older proof of stake algorithms have a "stake grinding" vulnerability where re-rolling the dice simply means making a computation locally on one's computer; algorithms with this vulnerability are certainly unacceptable in a sharding context. Newer algorithms (see the "validator selection" section in the [proof of stake FAQ]()) have the property that re-rolling the dice can only be done by voluntarily giving up one's spot in the block creation process, which entails giving up rewards and fees. The best way to mitigate the impact of marginal economically motivated attacks on sample selection is to find ways to increase this cost. One method to increase the cost by a factor of sqrt(N) from N rounds of voting is the [majority-bit method devised by Iddo Bentov]().

Another form of random number generation that is not exploitable by minority coalitions is the deterministic threshold signature approach most researched and advocated by Dominic Williams. The strategy here is to use a [deterministic threshold signature]() to generate the random seed from which samples are selected. Deterministic threshold signatures have the property that the value is guaranteed to be the same regardless of which of a given set of participants provides their data to the algorithm, provided that at least ⅔ of participants do participate honestly. This approach is more obviously not economically exploitable and fully resistant to all forms of stake-grinding, but it has several weaknesses:

- **It relies on more complex cryptography** (specifically, elliptic curves and pairings). Other approaches rely on nothing but the random-oracle assumption for common hash algorithms.
- **It fails when many validators are offline**. A desired goal for public blockchains is to be able to survive very large portions of the network simultaneously disappearing, as long as a majority of the remaining nodes is honest; deterministic threshold signature schemes at this point cannot provide this property.
- **It's not secure in a bribing attacker or coordinated majority model** where more than 67% of validators are colluding. The other approaches described in the proof of stake FAQ above still make it expensive to manipulate the randomness, as data from all validators is mixed into the seed and making any manipulation requires either universal collusion or excluding other validators outright.

One might argue that the deterministic threshold signature approach works better in consistency-favoring contexts and other approaches work better in availability-favoring contexts.

What are the tradeoffs in making sampling more or less frequent?

Selection frequency affects just how adaptive adversaries can be for the protocol to still be secure against them; for example, if you believe that an adaptive attack (e.g. dishonest validators who discover that they are part of the same sample banding together and colluding) can happen in 6 hours but not less, then you would be okay with a sampling time of 4 hours but not 12 hours. This is an argument in favor of making sampling happen as quickly as possible.

The main challenge with sampling taking place every block is that reshuffling carries a very high amount of overhead. Specifically, verifying a block on a shard requires knowing the state of that shard, and so every time validators are reshuffled, validators need to download the entire state for the new shard(s) that they are in. This requires both a strong state size control policy (i.e. economically ensuring that the size of the state does not grow too large, whether by deleting old accounts, restricting the rate of creating new accounts or a combination of the two) and a fairly long reshuffling time to work well.

Currently, the Parity client can download and verify a full Ethereum state snapshot via "warp-sync" in ~2-8 hours, suggesting that reshuffling periods of a few days but not less are safe; perhaps this could be reduced somewhat by shrinking the state size via storage rent but even still reshuffling periods would need to be long, potentially making the system vulnerable to adaptive adversaries.

However, there are ways of completely avoiding the tradeoff, choosing the creator of the next collation in each shard with only a few minutes of warning but without adding impossibly high state downloading overhead. This is done by shifting responsibility for state storage, and possibly even state execution, away from collators entirely, and instead assigning the role to either users or an interactive verification protocol.

Can we force more of the state to be held user-side so that transactions can be validated without requiring validators to hold all state data?

See also: https://ethresear.ch/t/the-stateless-client-concept/172

The techniques here tend to involve requiring users to store state data and provide Merkle proofs along with every transaction that they send. A transaction would be sent along with a Merkle proof-of-correct-execution (or "witness"), and this proof would allow a node that only has the state root to calculate the new state root. This proof-of-correct-execution would consist of the subset of objects in the trie that would need to be traversed to access and verify the state information that the transaction must verify; because Merkle proofs are O(log(n)) sized, the proof for a transaction that accesses a constant number of objects would also be O(log(n)) sized.
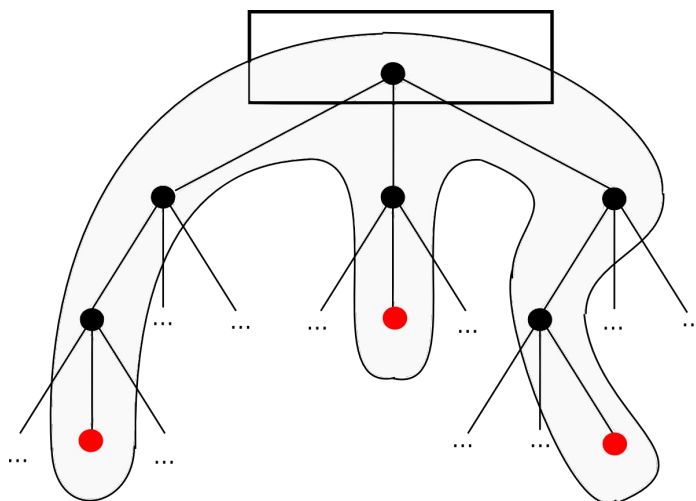
The subset of objects in a Merkle tree that would need to be provided in a Merkle proof of a transaction that accesses several state objects

Implementing this scheme in its pure form has two flaws. First, it introduces O(log(n)) overhead (~10-30x in practice), although one could argue that this O(log(n)) overhead is not as bad as it seems

because it ensures that the validator can always simply keep state data in memory and thus it never needs to deal with the overhead of accessing the hard drive[9]. Second, it can easily be applied if the addresses that are accessed by a transaction are static, but is more difficult to apply if the addresses in question are dynamic - that is, if the transaction execution has code of the form `read(f(read(x)))` where the address of some state read depends on the execution result of some other state read. In this case, the address that the transaction sender thinks the transaction will be reading at the time that they send the transaction may well differ from the address that is actually read when the transaction is included in a block, and so the Merkle proof may be insufficient[10].

This can be solved with access lists (think: a list of accounts and subsets of storage tries), which specify statically what data transactions can access, so when a miner receives a transaction with a witness they can determine that the witness contains all of the data the transaction could possibly access or modify. However, this harms censorship resistance, making attacks similar in form to the attempted DAO soft fork possible.

Can we split data and execution so that we get the security from rapid shuffling data validation without the overhead of shuffling the nodes that perform state execution?

Yes. We can create a protocol where we split up validators into three roles with different incentives (so that the incentives do not overlap): **proposers or collators, a.k.a. prolators**, **notaries** and **executors**. Prollators are responsible for simply building a chain of collations; while notaries verify that the data in the collations is available. Prolators do not need to verify anything state-dependent (e.g. whether or not someone trying to send ETH has enough money). Executors take the chain of collations agreed to by the prolators as given, and then execute the transactions in the collations sequentially and compute the state. If any transaction included in a collation is invalid, executors simply skip over it. This way, validators that verify availability could be reshuffled instantly, and executors could stay on one shard.

There would be a light client protocol that allows light clients to determine what the state is based on claims signed by executors, but this protocol is NOT a simple majority-voting consensus. Rather, the protocol is an interactive game with some similarities to Truebit, where if there is great disagreement then light client simply execute specific collations or portions of collations themselves. Hence, light clients can get a correct view of the state even if 90% of the executors in the shard are corrupted, making it much safer to allow executors to be very infrequently reshuffled or even permanently shard-specific.

Choosing *what goes in* to a collation does require knowing the state of that collation, as that is the most practical way to know what will actually pay transaction fees, but this can be solved by further separating the role of collators (who agree on the history) and proposers (who propose individual collations) and creating a market between the two classes of actors; see here for more discussion on this. However, this approach has since been found to be flawed as per this analysis.

Can SNARKs and STARKs help?

Yes! One can create a second-level protocol where a SNARK, STARK or similar succinct zero knowledge proof scheme is used to prove the state root of a shard chain, and proof creators can be rewarded for this. That said, shard chains to actually agree on what data gets included into the shard chains in the first place is still required.

How can we facilitate cross-shard communication?

The easiest scenario to satisfy is one where there are very many applications that individually do not have too many users, and which only very occasionally and loosely interact with each other; in this case, applications can live on separate shards and use cross-shard communication via receipts to talk to each other.

This typically involves breaking up each transaction into a "debit" and a "credit". For example, suppose that we have a transaction where account A on shard M wishes to send 100 coins to account B on shard N. The steps would looks as follows:

1. Send a transaction on shard M which (i) deducts the balance of A by 100 coins, and (ii) creates a receipt. A receipt is an object which is not saved in the state directly, but where the fact that the receipt was generated can be verified via a Merkle proof.
2. Wait for the first transaction to be included (sometimes waiting for finalization is required; this depends on the system).
3. Send a transaction on shard N which includes the Merkle proof of the receipt from (1). This transaction also checks in the state of shard N to make sure that this receipt is "unspent"; if it is, then it increases the balance of B by 100 coins, and saves in the state that the receipt is spent.
4. Optionally, the transaction in (3) also saves a receipt, which can then be used to perform further actions on shard M that are contingent on the original operation succeeding.
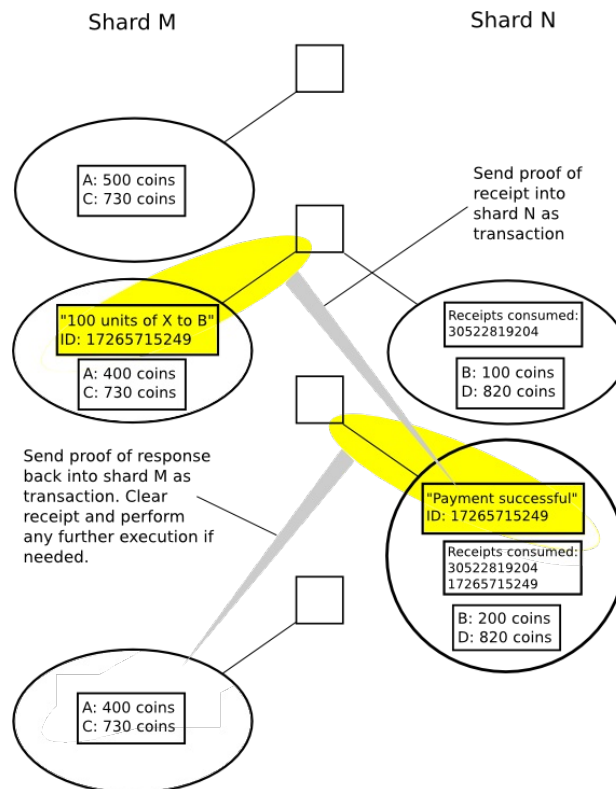
In more complex forms of sharding, transactions may in some cases have effects that spread out across several shards and may also synchronously ask for data from the state of multiple shards.

What is the train-and-hotel problem?

The following example is courtesy of Andrew Miller. Suppose that a user wants to purchase a train ticket and reserve a hotel, and wants to make sure that the operation is atomic - either both reservations succeed or neither do. If the train ticket and hotel booking applications are on the same shard, this is easy: create a transaction that attempts to make both reservations, and throws an exception and reverts everything unless both reservations succeed. If the two are on different shards, however, this is not so easy; even without cryptoeconomic / decentralization concerns, this is essentially the problem of atomic database transactions.

With asynchronous messages only, the simplest solution is to first reserve the train, then reserve the hotel, then once both reservations succeed confirm both; the reservation mechanism would prevent anyone else from reserving (or at least would ensure that enough spots are open to allow all reservations to be confirmed) for some period of time. However, this means that the mechanism relies on an extra security assumptions: that cross-shard messages from one shard can get included in another shard within some fixed period of time.

With cross-shard synchronous transactions, the problem is easier, but the challenge of creating a sharding solution capable of making cross-shard atomic synchronous transactions is itself decidedly nontrivial; see Vlad Zamfir's presentation which talks about merge blocks.

Another solution involves making contracts themselves movable across shards; see the proposed [cross-shard locking scheme](#) as well as [this proposal](#) where contracts can be "yanked" from one shard to another, allowing two contracts that normally reside on different shards to be temporarily moved to the same shard at which point a synchronous operation between them can happen.

What are the concerns about sharding through random sampling in a bribing attacker or coordinated choice model?

In a bribing attacker or coordinated choice model, the fact that validators are randomly sampled doesn't matter: whatever the sample is, either the attacker can bribe the great majority of the sample to do as the attacker pleases, or the attacker controls a majority of the sample directly and can direct the sample to perform arbitrary actions at low cost (O(c) cost, to be precise).

At that point, the attacker has the ability to conduct 51% attacks against that sample. The threat is further magnified because there is a risk of cross-shard contagion: if the attacker corrupts the state of a shard, the attacker can then start to send unlimited quantities of funds out to other shards and perform other cross-shard mischief. All in all, security in the bribing attacker or coordinated choice model is not much better than that of simply creating O(c) altcoins.

How can we improve on this?

In the context of state execution, we can use interactive verification protocols that are not randomly sampled majority votes, and that can give correct answers even if 90% of the participants are faulty; see [Truebit](#) for an example of how this can be done. For data availability, the problem is harder, though there are several strategies that can be used alongside majority votes to solve it.

What is the data availability problem, and how can we use erasure codes to solve it?

See https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding

Can we remove the need to solve data availability with some kind of fancy cryptographic accumulator scheme?

No. Suppose there is a scheme where there exists an object S representing the state (S could possibly be a hash) possibly as well as auxiliary information ("witnesses") held by individual users that can prove the presence of existing state objects (e.g. S is a Merkle root, the witnesses are the branches, though other constructions like RSA accumulators do exist). There exists an updating protocol where some data is broadcasted, and this data changes S to change the contents of the state, and also possibly changes witnesses.

Suppose some user has the witnesses for a set of N objects in the state, and M of the objects are updated. After receiving the update information, the user can check the new status of all N objects, and thereby see which M were updated. Hence, the update information itself encoded at least ~M * log(N) bits of information. Hence, the update information that everyone needs for receive to implement the effect of M transactions must necessarily be of size O(M). [14]

So this means that we can actually create scalable sharded blockchains where the cost of making anything bad happen is

proportional to the size of the entire validator set?

There is one trivial attack by which an attacker can always burn O(c) capital to temporarily reduce the quality of a shard: spam it by sending transactions with high transaction fees, forcing legitimate users to outbid you to get in. This attack is unavoidable; you could compensate with flexible gas limits, and you could even try "transparent sharding" schemes that try to automatically re-allocate nodes to shards based on usage, but if some particular application is non-parallelizable, Amdahl's law guarantees that there is nothing you can do. The attack that is opened up here (reminder: it only works in the Zamfir model, not honest/uncoordinated majority) is arguably not substantially worse than the transaction spam attack. Hence, we've reached the known limit for the security of a single shard, and there is no value in trying to go further.

Let's walk back a bit. Do we actually need any of this complexity if we have instant shuffling? Doesn't instant shuffling basically mean that each shard directly pulls validators from the global validator pool so it operates just like a blockchain, and so sharding doesn't actually introduce any new complexities?

Kind of. First of all, it's worth noting that proof of work and simple proof of stake, even without sharding, both have very low security in a bribing attacker model; a block is only truly "finalized" in the economic sense after O(n) time (as if only a few blocks have passed, then the economic cost of replacing the chain is simply the cost of starting a double-spend from before the block in question). Casper solves this problem by adding its finality mechanism, so that the economic security margin immediately increases to the maximum. In a sharded chain, if we want economic finality then we need to come up with a chain of reasoning for why a validator would be willing to make a very strong claim on a chain based solely on a random sample, when the validator itself is convinced that the bribing attacker and coordinated choice models may be true and so the random sample could potentially be corrupted.

You mentioned transparent sharding. I'm 12 years old and what is this?

Basically, we do not expose the concept of "shards" directly to developers, and do not permanently assign state objects to specific shards. Instead, the protocol has an ongoing built-in load-balancing process that shifts objects around between shards. If a shard gets too big or consumes too much gas it can be split in half; if two shards get too small and talk to each other very often they can be combined together; if all shards get too small one shard can be deleted and its contents moved to various other shards, etc.

Imagine if Donald Trump realized that people travel between New York and London a lot, but there's an ocean in the way, so he could just take out his scissors, cut out the ocean, glue the US east coast and Western Europe together and put the Atlantic beside the South Pole - it's kind of like that.

What are some advantages and disadvantages of this?

- Developers no longer need to think about shards

- There's the possibility for shards to adjust manually to changes in gas prices, rather than relying on market mechanics to increase gas prices in some shards more than others
- There is no longer a notion of reliable co-placement: if two contracts are put into the same shard so that they can interact with each other, shard changes may well end up separating them
- More protocol complexity

The co-placement problem can be mitigated by introducing a notion of "sequential domains", where contracts may specify that they exist in the same sequential domain, in which case synchronous communication between them will always be possible. In this model a shard can be viewed as a set of sequential domains that are validated together, and where sequential domains can be rebalanced between shards if the protocol determines that it is efficient to do so.

How would synchronous cross-shard messages work?

The process becomes much easier if you view the transaction history as being already settled, and are simply trying to calculate the state transition function. There are several approaches; one fairly simple approach can be described as follows:

- A transaction may specify a set of shards that it can operate in
- In order for the transaction to be effective, it must be included at the same block height in all of these shards.
- Transactions within a block must be put in order of their hash (this ensures a canonical order of execution)

A client on shard X, if it sees a transaction with shards (X, Y), requests a Merkle proof from shard Y verifying (i) the presence of that transaction on shard Y, and (ii) what the pre-state on shard Y is for those bits of data that the transaction will need to access. It then executes the transaction and commits to the execution result. Note that this process may be highly inefficient if there are many transactions with many different "block pairings" in each block; for this reason, it may be optimal to simply require blocks to specify sister shards, and then calculation can be done more efficiently at a per-block level. This is the basis for how such a scheme could work; one could imagine more complex designs. However, when making a new design, it's always important to make sure that low-cost denial of service attacks cannot arbitrarily slow state calculation down.

What about semi-asynchronous messages?

Vlad Zamfir created a scheme by which asynchronous messages could still solve the "book a train and hotel" problem. This works as follows. The state keeps track of all operations that have been recently made, as well as the graph of which operations were triggered by any given operation (including cross-shard operations). If an operation is reverted, then a receipt is created which can then be used to revert any effect of that operation on other shards; those reverts may then trigger their own reverts and so forth. The argument is that if one biases the system so that revert messages can propagate twice as fast as other kinds of messages, then a complex cross-shard transaction that finishes executing in K rounds can be fully reverted in another K rounds.

The overhead that this scheme would introduce has arguably not been sufficiently studied; there may be worst-case scenarios that trigger quadratic execution vulnerabilities. It is clear that if transactions have effects that are more isolated from each other, the overhead of this mechanism is lower; perhaps isolated executions can be incentivized via favorable gas cost rules. All in all, this is one of the more promising research directions for advanced sharding.

What are guaranteed cross-shard calls?

One of the challenges in sharding is that when a call is made, there is by default no hard protocol-provided guarantee that any asynchronous operations created by that call will be made within any particular timeframe, or even made at all; rather, it is up to some party to send a transaction in the destination shard triggering the receipt. This is okay for many applications, but in some cases it may be problematic for several reasons:

- There may be no single party that is clearly incentivized to trigger a given receipt. If the sending of a transaction benefits many parties, then there could be **tragedy-of-the-commons effects**

where the parties try to wait longer until someone else sends the transaction (i.e. play "chicken"), or simply decide that sending the transaction is not worth the transaction fees for them individually.

- **Gas prices across shards may be volatile**, and in some cases performing the first half of an operation compels the user to "follow through" on it, but the user may have to end up following through at a much higher gas price. This may be exacerbated by DoS attacks and related forms of **griefing**.
- Some applications rely on there being an upper bound on the "latency" of cross-shard messages (e.g. the train-and-hotel example). Lacking hard guarantees, such applications would have to have **inefficiently large safety margins**.

One could try to come up with a system where asynchronous messages made in some shard automatically trigger effects in their destination shard after some number of blocks. However, this requires every client on each shard to actively inspect all other shards in the process of calculating the state transition function, which is arguably a source of inefficiency. The best known compromise approach is this: when a receipt from shard A at height `height_a` is included in shard B at height `height_b`, if the difference in block heights exceeds `MAX_HEIGHT`, then all validators in shard B that created blocks from `height_a + MAX_HEIGHT + 1` to `height_b - 1` are penalized, and this penalty increases exponentially. A portion of these penalties is given to the validator that finally includes the block as a reward. This keeps the state transition function simple, while still strongly incentivizing the correct behavior.

Wait, but what if an attacker sends a cross-shard call from every shard into shard X at the same time? Wouldn't it be mathematically impossible to include all of these calls in time?

Correct; this is a problem. Here is a proposed solution. In order to make a cross-shard call from shard A to shard B, the caller must pre-purchase "congealed shard B gas" (this is done via a transaction in shard B, and recorded in shard B). Congealed shard B gas has a fast demurrage rate: once ordered, it loses 1/k of its remaining potency every block. A transaction on shard A can then send the congealed shard B gas along with the receipt that it creates, and it can be used on shard B for free. Shard B blocks allocate extra gas space specifically for these kinds of transactions. Note that because of the demurrage rules, there can be at most GAS_LIMIT * k worth of congealed gas for a given shard available at any time, which can certainly be filled within k blocks (in fact, even faster due to demurrage, but we may need this slack space due to malicious validators). In case too many validators maliciously fail to include receipts, we can make the penalties fairer by exempting validators who fill up the "receipt space" of their blocks with as many receipts as possible, starting with the oldest ones.

Under this pre-purchase mechanism, a user that wants to perform a cross-shard operation would first pre-purchase gas for all shards that the operation would go into, over-purchasing to take into account the demurrage. If the operation would create a receipt that triggers an operation that consumes 100000 gas in shard B, the user would pre-buy 100000 * e (i.e. 271818) shard-B congealed gas. If that operation would in turn spend 100000 gas in shard C (i.e. two levels of indirection), the user would need to pre-buy 100000 * e^2 (i.e. 738906) shard-C congealed gas. Notice how once the purchases are confirmed, and the user starts the main operation, the user can be confident that they will be insulated from changes in the gas price market, unless validators voluntarily lose large quantities of money from receipt non-inclusion penalties.

Congealed gas? This sounds interesting for not just cross-shard operations, but also reliable intra-shard scheduling

Indeed; you could buy congealed shard A gas inside of shard A, and send a guaranteed cross-shard call from shard A to itself. Though note that this scheme would only support scheduling at very short time intervals, and the scheduling would not be exact to the block; it would only be guaranteed to

happen within some period of time.

Does guaranteed scheduling, both intra-shard and cross-shard, help against majority collusions trying to censor transactions?

Yes. If a user fails to get a transaction in because colluding validators are filtering the transaction and not accepting any blocks that include it, then the user could send a series of messages which trigger a chain of guaranteed scheduled messages, the last of which reconstructs the transaction inside of the EVM and executes it. Preventing such circumvention techniques is practically impossible without shutting down the guaranteed scheduling feature outright and greatly restricting the entire protocol, and so malicious validators would not be able to do it easily.

Could sharded blockchains do a better job of dealing with network partitions?

The schemes described in this document would offer no improvement over non-sharded blockchains; realistically, every shard would end up with some nodes on both sides of the partition. There have been calls (e.g. from IPFS's Juan Benet) for building scalable networks with the specific goal that networks can split up into shards as needed and thus continue operating as much as possible under network partition conditions, but there are nontrivial cryptoeconomic challenges in making this work well.

One major challenge is that if we want to have location-based sharding so that geographic network partitions minimally hinder intra-shard cohesion (with the side effect of having very low intra-shard latencies and hence very fast intra-shard block times), then we need to have a way for validators to choose which shards they are participating in. This is dangerous, because it allows for much larger classes of attacks in the honest/uncoordinated majority model, and hence cheaper attacks with higher griefing factors in the Zamfir model. Sharding for geographic partition safety and sharding via random sampling for efficiency are two fundamentally different things.

Second, more thinking would need to go into how applications are organized. A likely model in a sharded blockchain as described above is for each "app" to be on some shard (at least for small-scale apps); however, if we want the apps themselves to be partition-resistant, then it means that all apps would need to be cross-shard to some extent.

One possible route to solving this is to create a platform that offers both kinds of shards - some shards would be higher-security "global" shards that are randomly sampled, and other shards would be lower-security "local" shards that could have properties such as ultra-fast block times and cheaper transaction fees. Very low-security shards could even be used for data-publishing and messaging.

What are the unique challenges of pushing scaling past n = O(c^2)?

There are several considerations. First, the algorithm would need to be converted from a two-layer algorithm to a stackable n-layer algorithm; this is possible, but is complex. Second, n / c (i.e. the ratio between the total computation load of the network and the capacity of one node) is a value that happens to be close to two constants: first, if measured in blocks, a timespan of several hours, which is an acceptable "maximum security confirmation time", and second, the ratio between rewards and deposits (an early computation suggests a 32 ETH deposit size and a 0.05 ETH block reward for Casper). The latter has the consequence that if rewards and penalties on a shard are escalated to be on the scale of validator deposits, the cost of continuing an attack on a shard will be O(n) in size.

Going above c^2 would likely entail further weakening the kinds of security guarantees that a system can provide, and allowing attackers to attack individual shards in certain ways for extended periods of time at medium cost, although it should still be possible to prevent invalid state from being finalized and to prevent finalized state from being reverted unless attackers are willing to pay an O(n) cost. However, the rewards are large - a super-quadratically sharded blockchain could be used as a general-purpose tool for nearly all decentralized applications, and could sustain transaction fees

that makes its use virtually free.

What about heterogeneous sharding?

Abstracting the execution engine or allowing multiple execution engines to exist results in being able to have a different execution engine for each shard. Due to Casper CBC being able to explore the full [tradeoff triangle](#), it is possible to alter the parameters of the consensus engine for each shard to be at any point of the triangle. However, CBC Casper has not been implemented yet, and heterogeneous sharding is nothing more than an idea at this stage; the specifics of how it would work has not been designed nor implemented. Some shards could be optimized to have fast finality and high throughput, which is important for applications such as EFTPOS transactions, while maybe most could have a moderate or reasonable amount each of finality, throughput and decentralization (number of validating nodes), and applications that are prone to a high fault rate and thus require high security, such as torrent networks, privacy focused email like Proton mail, etc., could optimize for a high decentralization, low finality and high throughput, etc. See also https://twitter.com/VladZamfir/status/932320997021171712 and https://ethresear.ch/t/heterogeneous-sharding/1979/2.

Footnotes

1. Merklix tree == Merkle Patricia tree

2. Later proposals from the NUS group do manage to shard state; they do this via the receipt and state-compacting techniques that I describe in later sections in this document. (This is Vitalik Buterin writing as the creator of this Wiki.)

3. There are reasons to be conservative here. Particularly, note that if an attacker comes up with worst-case transactions whose ratio between processing time and block space expenditure (bytes, gas, etc) is much higher than usual, then the system will experience very low performance, and so a safety factor is necessary to account for this possibility. In traditional blockchains, the fact that block processing only takes ~1-5% of block time has the primary role of protecting against centralization risk but serves double duty of protecting against denial of service risk. In the specific case of Bitcoin, its current worst-case [known quadratic execution vulnerability](#) arguably limits any scaling at present to ~5-10x, and in the case of Ethereum, while all known vulnerabilities are being or have been removed after the denial-of-service attacks, there is still a risk of further discrepancies particularly on a smaller scale. In Bitcoin NG, the need for the former is removed, but the need for the latter is still there.

4. A further reason to be cautious is that increased state size corresponds to reduced throughput, as nodes will find it harder and harder to keep state data in RAM and so need more and more disk accesses, and databases, which often have an $O(\log(n))$ access time, will take longer and longer to access. This was an important lesson from the last Ethereum denial-of-service attack, which bloated the state by ~10 GB by creating empty accounts and thereby indirectly slowed processing down by forcing further state accesses to hit disk instead of RAM.

5. In sharded blockchains, there may not necessarily be in-lockstep consensus on a single global state, and so the protocol never asks nodes to try to compute a global state root; in fact, in the protocols presented in later sections, each shard has its own state, and for each shard there is a mechanism for committing to the state root for that shard, which represents that shard's state

6. #MEGA

7. If a non-scalable blockchain upgrades into a scalable blockchain, the author's recommended path is that the old chain's state should simply become a single shard in the new chain.

8. For this to be secure, some further conditions must be satisfied; particularly, the proof of work must be non-outsourceable in order to prevent the attacker from determining which *other miners' identities* are available for some given shard and mining on top of those.

9. Recent Ethereum denial-of-service attacks have proven that hard drive access is a primary bottleneck to blockchain scalability.

10. You could ask: well why don't validators fetch Merkle proofs just-in-time? Answer: because doing so is a ~100-1000ms roundtrip, and executing an entire complex transaction within that time could be prohibitive.

11. One hybrid solution that combines the normal-case efficiency of small samples with the greater robustness of larger samples is a multi-layered sampling scheme: have a consensus between 50 nodes that requires 80% agreement to move forward, and then only if that consensus fails to be reached then fall back to a 250-node sample. N = 50 with an 80% threshold has only a 8.92 * 10-9 failure rate even against attackers with p = 0.4, so this does not harm security at all under an honest or uncoordinated majority model.

12. The probabilities given are for one single shard; however, the random seed affects O(c) shards and the attacker could potentially take over any one of them. If we want to look at O(c) shards simultaneously, then there are two cases. First, if the grinding process is computationally bounded, then this fact does not change the calculus at all, as even though there are now O(c) chances of success per round, checking success takes O(c) times as much work. Second, if the grinding process is economically bounded, then this indeed calls for somewhat higher safety factors (increasing N by 10-20 should be sufficient) although it's important to note that the goal of an attacker in a profit-motivated manipulation attack is to increase their participation across all shards in any case, and so that is the case that we are already investigating.

13. See [Parity's Polkadotpaper](#) for further description of how their "fishermen" concept works. For up-to-date info and code for Polkadot, see [here](#).

14. Thanks to Justin Drake for pointing me to cryptographic accumulators, as well as [this paper](#) that gives the argument for the impossibility of sublinear batching. See also this thread: https://ethresear.ch/t/accumulators-scalability-of-utxo-blockchains-and-data-availability/176

Further reading related to sharding, and more generally scalability and research, is available [here](#) and [here](#).

Notes on Blockchain Governance

2017 Dec 17

In which I argue that "tightly coupled" on-chain voting is overrated, the status quo of "informal governance" as practiced by Bitcoin, Bitcoin Cash, Ethereum, Zcash and similar systems is much less bad than commonly thought, that people who think that the purpose of blockchains is to completely expunge soft mushy human intuitions and feelings in favor of completely algorithmic governance (emphasis on "completely") are absolutely crazy, and loosely coupled voting as done by Carbonvotes and similar systems is underrated, as well as describe what framework should be used when thinking about blockchain governance in the first place.

See also: https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca

One of the more interesting recent trends in blockchain governance is the resurgence of on-chain coin-holder voting as a multi-purpose decision mechanism. Votes by coin holders are sometimes used in order to decide who operates the super-nodes that run a network (eg. DPOS in EOS, NEO, Lisk and other systems), sometimes to vote on protocol parameters (eg. the Ethereum gas limit) and sometimes to vote on and directly implement protocol upgrades wholesale (eg. Tezos). In all of these cases, the votes are automatic - the protocol itself contains all of the logic needed to change the validator set or to update its own rules, and does this automatically in response to the result of votes.

Explicit on-chain governance is typically touted as having several major advantages. First, unlike the highly conservative philosophy espoused by Bitcoin, it can evolve rapidly and accept needed technical improvements. Second, by creating an *explicit* decentralized framework, it avoids the perceived pitfalls of *informal* governance, which is viewed to either be too unstable and prone to chain splits, or prone to becoming too de-facto centralized - the latter being the same argument made in the famous 1972 essay "Tyranny of Structurelessness".

Quoting Tezos documentation:

> While all blockchains offer financial incentives for maintaining consensus on their ledgers, no blockchain has a robust on-chain mechanism that seamlessly amends the rules governing its protocol and rewards protocol development. As a result, first-generation blockchains empower de facto, centralized core development teams or miners to formulate design choices.

And:

> Yes, but why would you want to make [a minority chain split] easier? Splits destroy network effects.

On-chain governance used to select validators also has the benefit that it allows for networks that impose high computational performance requirements on validators without introducing economic centralization risks and other traps of the kind that appear in public blockchains (eg. the validator's dilemma).

So far, all in all, on-chain governance seems like a very good bargain.... so what's wrong with it?

What is Blockchain Governance?

To start off, we need to describe more clearly what the process of "blockchain governance" *is*. Generally speaking, there are two informal models of governance, that I will call the "decision function" view of governance and the "coordination" view of governance. The decision function view treats governance as a function $f(x_1, x_2 ... x_n) \rightarrow y$, where the inputs are the wishes of various legitimate stakeholders (senators, the president, property owners, shareholders, voters, etc) and the output is the decision.

Inputs

x_1
x_2
x_3
x_4
x_5
x_6

Decision function

Output
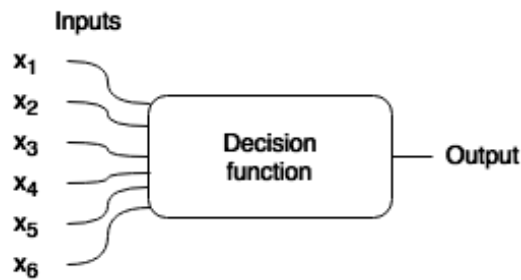
The decision function view is often useful as an approximation, but it clearly frays very easily around the edges: people often can and do break the law and get away with it, sometimes rules are ambiguous, and sometimes revolutions happen - and all three of these possibilities are, at least sometimes, *a good thing*. And often even behavior inside the system is shaped by incentives created by *the possibility* of acting outside the system, and this once again is at least sometimes a good thing.

The coordination model of governance, in contrast, sees governance as something that exists in layers. The bottom layer is, in the real world, the laws of physics themselves (as a geopolitical realist would say, guns and bombs), and in the blockchain space we can abstract a bit further and say that it is each individual's ability to run whatever software they want in their capacity as a user, miner, stakeholder, validator or whatever other kind of agent a blockchain protocol allows them to be. The bottom layer is always the ultimate deciding layer; if, for example, all Bitcoin users wake up one day and decides to edit their clients' source code and replace the entire code with an Ethereum client that listens to balances of a particular ERC20 token contract, then that means that that ERC20 token *is* bitcoin. The bottom layer's ultimate governing power cannot be stopped, but the actions that people take on this layer can be *influenced* by the layers above it.

The second (and crucially important) layer is coordination institutions. The purpose of a coordination institution is to create focal points around how and when individuals should act in order to better coordinate behavior. There are many situations, both in blockchain governance and in real life, where if you act in a certain way alone, you are likely to get nowhere (or worse), but if everyone acts together a desired result can be achieved.

	A	B
A	(5, 5)	(0, 0)
B	(0, 0)	(5, 5)

An abstract coordination game. You benefit heavily from making the same move as everyone else.

In these cases, it's in your interest to go if everyone else is going, and stop if everyone else is stopping. You can think of coordination institutions as putting up green or red flags in the air saying "go" or "stop", *with an established culture* that everyone watches these flags and (usually) does what they say. Why do people have the incentive to follow these flags? Because *everyone else* is already following these flags, and you have the incentive to do the same thing as what everyone else is doing.

A Byzantine general rallying his troops forward. The purpose of this isn't just to make the soldiers feel brave and excited, but also to reassure them that *everyone else* feels brave and excited and will charge forward as well, so an individual soldier is not just committing suicide by charging forward alone.

Strong claim: this concept of coordination flags encompasses *all* that we mean by "governance"; in scenarios where coordination games (or more generally, multi-equilibrium games) do not exist, the concept of governance is meaningless.

In the real world, military orders from a general function as a flag, and in the blockchain world, the simplest example of such a flag is the mechanism that tells people whether or not a hard fork "is happening". Coordination institutions can be very formal, or they can be informal, and often give suggestions that are ambiguous. Flags would ideally always be either red or green, but sometimes a flag might be yellow, or even holographic, appearing green to some participants and yellow or red to others. Sometimes that are also multiple flags that conflict with each other.

The key questions of governance thus become:

- What should layer 1 be? That is, what features should be set up in the initial protocol itself, and how does this influence the ability to make formulaic (ie. decision-function-like) protocol changes, as well as the level of power of different kinds of agents to act in different ways?
- What should layer 2 be? That is, what coordination institutions should people be encouraged to care about?

The Role of Coin Voting

Ethereum also has a history with coin voting, including:

- **DAO proposal votes**: https://daostats.github.io/proposals.html
- **The DAO Carbonvote**: http://v1.carbonvote.com/
- **The EIP 186/649/669 Carbonvote**: http://carbonvote.com/

Vote Status

■ YES ■ NO
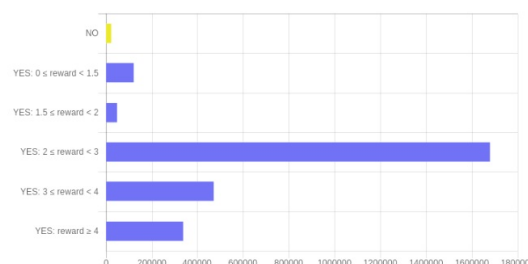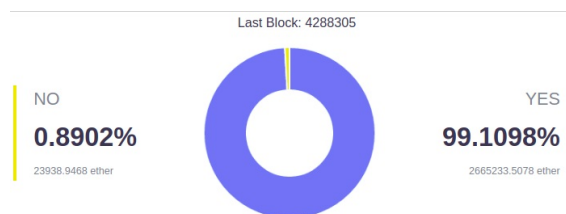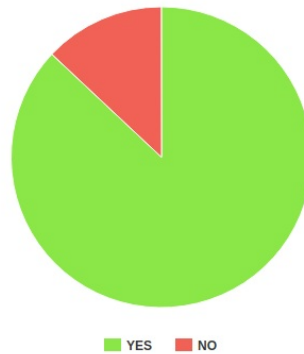
Open Proposals   Hide Splits

Number of open proposals: 71

ID	Description	Deposit	Time Left	Turnout	
2	Do you believe in god?	2 ether	Finished	0.75%	
71	beer split	0 ether	Finished	0%	Split
5	Moratorium on proposals until the DAO contract is ...	2 ether	Finished	8.13%	
11	Curators, please hire somebody to fix the DAO code...	2 ether	Finished	1.77%	
15	Dear DAO - Tokenholders, I am a simple DAO-Tokenho...	2 ether	Finished	2.24%	
17	Raising the Proposal Deposit to 11 ETH \n This P...	2 ether	Finished	9.62%	

These three are all examples of *loosely coupled* coin voting, or coin voting as a layer 2 coordination institution. Ethereum does not have any examples of *tightly coupled* coin voting (or, coin voting as a layer 1 in-protocol feature), though it *does* have an example of tightly coupled *miner* voting: miners' right to vote on the gas limit. Clearly, tightly coupled voting and loosely coupled voting are competitors in the governance mechanism space, so it's worth dissecting: what are the advantages and disadvantages of each one?

Assuming zero transaction costs, and if used as a sole governance mechanism, the two are clearly equivalent. If a loosely coupled vote says that change X should be implemented, then that will serve as a "green flag" encouraging everyone to download the update; if a minority wants to rebel, they will simply not download the update. If a tightly coupled vote implements change X, then the change happens automatically, and if a minority wants to rebel they can install a hard fork update that cancels the change. However, there clearly are nonzero transaction costs associated with making a hard fork, and this leads to some very important differences.

One very simple, and important, difference is that tightly coupled voting creates a default in favor of the blockchain adopting what the majority wants, requiring minorities to exert great effort to coordinate a hard fork to preserve a blockchain's existing properties, whereas loosely coupled voting is only a coordination tool, and still requires users to actually download and run the software that implements any given fork. But there are also many other differences. Now, let us go through some arguments *against* voting, and dissect how each argument applies to voting as layer 1 and voting as layer 2.

Low Voter Participation

One of the main criticisms of coin voting mechanisms so far is that, no matter where they are tried, they tend to have very low voter participation. The DAO Carbonvote only had a voter participation rate of 4.5%:
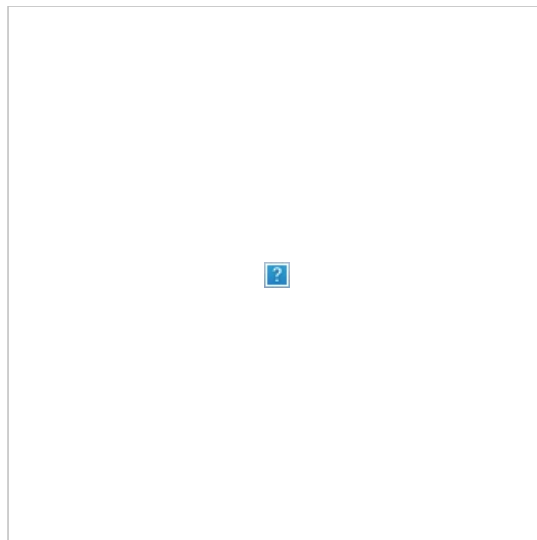
Additionally, wealth distribution is very unequal, and the results of these two factors together are best described by this image created by a critic of the DAO fork:
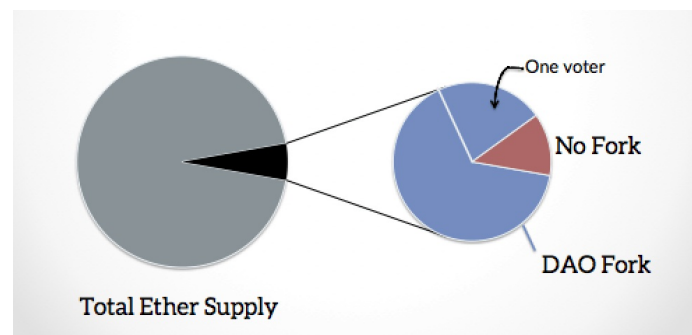
The EIP 186 Carbonvote had ~2.7 million ETH voting. The DAO proposal votes did not fare better, with participation never reaching 10%. And outside of Ethereum things are not sunny either; even in Bitshares, a system where the core social contract is designed around voting, the top delegate in an approval vote only got 17% of the vote, and in Lisk it got up to 30%, though as we will discuss later these systems have other problems of their own.

Low voter participation means two things. First, the vote has a harder time achieving a perception of legitimacy, because it only reflects the views of a small percentage of people. Second, an attacker with only a small percentage of all coins can sway the vote. These problems exist regardless of whether the vote is tightly coupled or loosely coupled.

Game-Theoretic Attacks

Aside from "the big hack" that received the bulk of the media attention, the DAO also had a number of much smaller game-theoretic vulnerabilities; this article from HackingDistributed does a good job of summarizing them. But this is only the tip of the iceberg. Even if all of the finer details of a voting mechanism are implemented correctly, voting mechanisms in general have a large flaw: in any vote, the probability that any given voter will have an impact on the result is tiny, and so the personal incentive that each voter has to vote correctly is almost insignificant. And if each person's size of the stake is small, their incentive to vote correctly is insignificant *squared*. Hence, a relatively small bribe spread out across the participants may suffice to sway their decision, possibly in a way that they collectively might quite disapprove of.
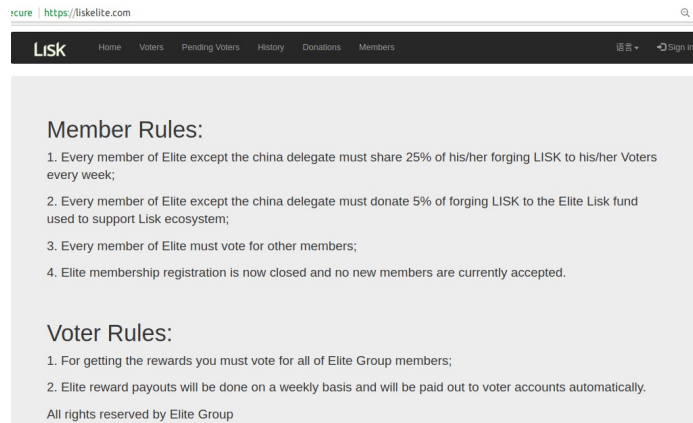
Now you might say, people are not evil selfish profit-maximizers that will accept a $0.5 bribe to vote to give twenty million dollars to Josh arza just because the above calculation says their individual chance of affecting anything is tiny; rather, they would altruistically refuse to do something that evil. There are two responses to this criticism.

First, there are ways to make a "bribe" that are quite plausible; for example, an exchange can offer

interest rates for deposits (or, even more ambiguously, use the exchange's own money to build a great interface and features), with the exchange operator using the large quantity of deposits to vote as they wish. Exchanges profit from chaos, so their incentives are clearly quite misaligned with users *and* coin holders.

Second, and more damningly, in practice it seems like people, at least in their capacity as crypto token holders, *are* profit maximizers, and seem to see nothing evil or selfish about taking a bribe or two. As "Exhibit A", we can look at the situation with Lisk, where the delegate pool seems to have been successfully captured by two major "political parties" that explicitly bribe coin holders to vote for them, and also require each member in the pool to vote for all the others.

Here's LiskElite, with 55 members (out of a total 101):

Here's LiskGDT, with 33 members:

And as "Exhibit B" some voter bribes being paid out in Ark:

Latest Transactions

| Id	Timestamp	Sender	Recipient	Smartbridge	Amount (ARK)	Fee (ARK)
380af...d7ab4	2017/04/17 12:20:41	bioly	AbxqF...jXJ6B	Payout from bioly delegate pool, thank you for support!	7.60466706	0.1
5795e...26029	2017/04/17 12:20:41	bioly	ARUNS...oLzvs	Payout from bioly delegate pool, thank you for support!	6.07691376	0.1
37694...35419	2017/04/17 12:20:40	bioly	AG2N1...taeZv	Payout from bioly delegate pool, thank you for support!	2.48455539	0.1
8c6b1...f1f9a	2017/04/17 12:20:39	bioly	AWmMj...HJU8R	Payout from bioly delegate pool, thank you for support!	118.47841646	0.1
d2ad5...c84af	2017/04/17 12:20:38	bioly	AbJ6N...ZrZxq	Payout from bioly delegate pool, thank you for support!	9.37653981	0.1
45280...aa3f0	2017/04/17 12:20:37	bioly	AevZb...68d6G	Payout from bioly delegate pool, thank you for support!	118.4945548	0.1
ace28...1cdee	2017/04/17 12:20:37	bioly	teletobi	Payout from bioly delegate pool, thank you for support!	11.72867675	0.1
20ca3...4278b	2017/04/17 12:20:36	bioly	ANY7W...6TfzX	Payout from bioly delegate pool, thank you for support!	4.80016674	0.1
a4de1...f90fd	2017/04/17 12:20:36	bioly	ARK8b...znv2Z	Payout from bioly delegate pool, thank you for support!	178.80073745	0.1
cb528...592bc	2017/04/17 12:20:36	bioly	AUmaL...QeHyP	Payout from bioly delegate pool, thank you for support!	237.32335576	0.1
29740...578db	2017/04/17 12:20:35	bioly	AUw4A...HxWB7	Payout from bioly delegate pool, thank you for support!	54.14948207	0.1
331df...5b0f2	2017/04/17 12:20:35	bioly	AQxnW...F2HGH	Payout from bioly delegate pool, thank you for support!	46.96456749	0.1
38fac...e02f5	2017/04/17 12:20:34	bioly	AKkvf...L5TW9	Payout from bioly delegate pool, thank you for support!	41.98709123	0.1
50190...b5284	2017/04/17 12:20:34	bioly	AWskK...bRB4m	Payout from bioly delegate pool, thank you for support!	7.39663982	0.1
72770...78a41	2017/04/17 12:20:34	bioly	AUTPB...E6pro	Payout from bioly delegate pool, thank you for support!	15.64031609	0.1
199f4...bae6a	2017/04/17 12:20:33	bioly	AVbiK...MEK8P	bioly fee account	403.66128558	0.1
af13f...6a16e	2017/04/17 12:20:33	bioly	AVVVY...gTtCo	Payout from bioly delegate pool, thank you for support!	7.63884129	0.1
74c3c...061f6	2017/04/17 12:20:32	bioly	AYTAy...3egy6	Payout from bioly delegate pool, thank you for support!	71.46381847	0.1

Here, note that there is a key difference between tightly coupled and loosely coupled votes. In a loosely coupled vote, direct or indirect vote bribing is also possible, but if the community agrees that some given proposal or set of votes constitutes a game-theoretic attack, they can simply socially agree to ignore it. And in fact this has kind of already happened - the Carbonvote contains a blacklist of addresses corresponding to known exchange addresses, and votes from these addresses are not counted. In a tightly coupled vote, there is no way to create such a blacklist at protocol level, because agreeing who is part of the blacklist is *itself* a blockchain governance decision. But since the blacklist is part of a community-created voting tool that only indirectly influences protocol changes, voting tools that contain bad blacklists can simply be rejected by the community.

It's worth noting that this section **is not** a prediction that all tightly coupled voting systems will quickly succumb to bribe attacks. It's entirely possible that many will survive for one simple reason: all of these projects have founders or foundations with large premines, and these act as large centralized actors that are interested in their platforms' success that are not vulnerable to bribes, and hold enough coins to outweigh most bribe attacks. However, this kind of centralized trust model, while arguably useful in some contexts in a project's early stages, is clearly one that is not sustainable in the long term.

Non-Representativeness

Another important objection to voting is that coin holders are only one class of user, and may have interests that collide with those of other users. In the case of pure cryptocurrencies like Bitcoin, store-of-value use ("hodling") and medium-of-exchange use ("buying coffees") are naturally in conflict, as the store-of-value prizes security much more than the medium-of-exchange use case, which more strongly values usability. With Ethereum, the conflict is worse, as there are many people who use Ethereum for reasons that have nothing to do with ether (see: cryptokitties), or even value-bearing digital assets in general (see: ENS).

Additionally, even if coin holders *are* the only relevant class of user (one might imagine this to be the case in a cryptocurrency where there is an established social contract that its purpose is to be the next digital gold, and nothing else), there is still the challenge that a coin holder vote gives a much greater voice to wealthy coin holders than to everyone else, opening the door for centralization of holdings to lead to unencumbered centralization of decision making. Or, in other words...
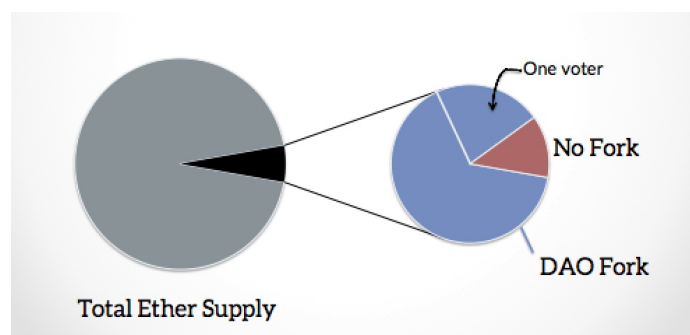
And if you want to see a review of a project that seems to combine all of these disadvantages at the same time, see this: https://btcgeek.com/bitshares-trying-memorycoin-year-ago-disastrous-ends/.

This criticism applies to both tightly coupled and loosely coupled voting equally; however, loosely coupled voting is more amenable to compromises that mitigate its unrepresentativeness, and we will discuss this more later.

Centralization

Let's look at the existing live experiment that we have in tightly coupled voting on Ethereum, the gas limit. Here's the gas limit evolution over the past two years:
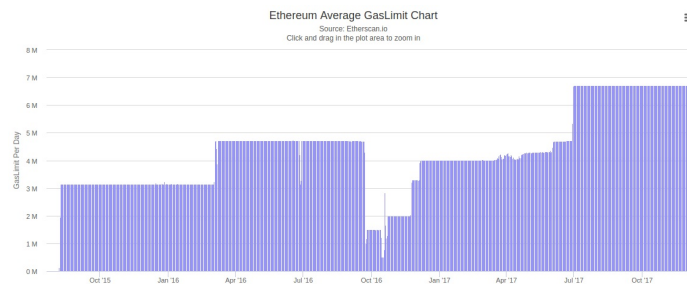
You might notice that the general feel of the curve is a bit like another chart that may be quite familiar to you:
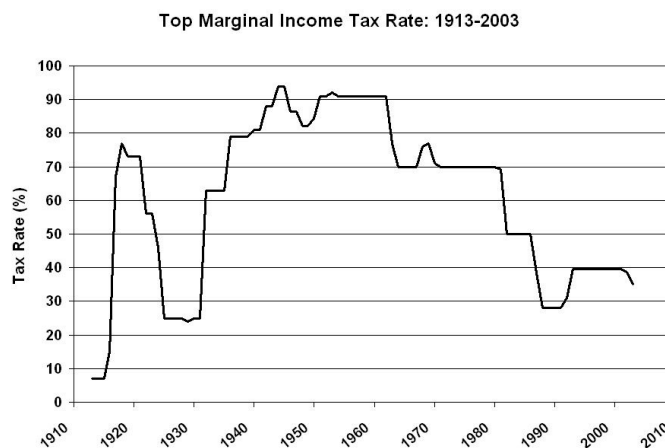
Basically, they both look like magic numbers that are created and repeatedly renegotiated by a fairly centralized group of guys sitting together in a room. What's happening in the first case? Miners are generally following the direction favored by the community, which is itself gauged via social consensus aids similar to those that drive hard forks (core developer support, Reddit upvotes, etc; in Ethereum, the gas limit has never gotten controversial enough to require anything as serious as a coin vote).

Hence, it is not at all clear that voting will be able to deliver *results* that are actually decentralized, if voters are not technically knowledgeable and simply defer to a single dominant tribe of experts. This criticism once again applies to tightly coupled and loosely coupled voting equally.

Update: since writing this, it seems like Ethereum miners managed to up the gas limit from 6.7 million to 8 million all without even discussing it with the core developers or the Ethereum Foundation. So there is hope; but it takes a lot of hard community building and other grueling non-technical work to get to that point.

Digital Constitutions

One approach that has been suggested to mitigate the risk of runaway bad governance algorithms is "digital constitutions" that mathematically specify desired properties that the protocol should have,

and require any new code changes to come with a computer-verifiable proof that they satisfy these properties. This seems like a good idea at first, but this too should, in my opinion, be viewed skeptically.

In general, the idea of having norms about protocol properties, and having these norms serve the function of one of the coordination flags, is a very good one. This allows us to enshrine core properties of a protocol that we consider to be very important and valuable, and make them more difficult to change. However, this is exactly the sort of thing that should be enforced in loosely coupled (ie. layer two), rather than tightly coupled (layer one) form.

Basically any meaningful norm is actually quite hard to express in its entirety; this is part of the [complexity of value](#) problem. This is true even for something as seemingly unambiguous as the 21 million coin limit. Sure, one can add a line of code saying `assert total_supply <= 21000000`, and put a comment around it saying "do not remove at all costs", but there are plenty of roundabout ways of doing the same thing. For example, one could imagine a soft fork that adds a mandatory transaction fee this is proportional to coin value * time since the coins were last sent, and this is equivalent to demurrage, which is equivalent to deflation. One could also implement another currency, called Bjtcoin, with 21 million *new* units, and add a feature where if a bitcoin transaction is sent the miner can intercept it and claim the bitcoin, instead giving the recipient bjtcoin; this would rapidly force bitcoins and bjtcoins to be fungible with each other, increasing the "total supply" to 42 million without ever tripping up that line of code. "Softer" norms like not interfering with application state are even harder to enforce.

We *want* to be able to say that a protocol change that violates any of these guarantees should be viewed as illegitimate - there should be a coordination institution that waves a red flag - even if they get approved by a vote. We also want to be able to say that a protocol change that follows the letter of a norm, but blatantly violates its spirit, the protocol change should *still* be viewed as illegitimate. And having norms exist on layer 2 - in the minds of humans in the community, rather than in the code of the protocol - best achieves that goal.

Toward A Balance

However, I am also not willing to go the other way and say that coin voting, or other explicit on-chain voting-like schemes, have no place in governance whatsoever. The leading alternative seems to be core developer consensus, however the failure mode of a system being controlled by "ivory tower intellectuals" who care more about abstract philosophies and solutions that sound technically impressive over and above real day-to-day concerns like user experience and transaction fees is, in my view, also a real threat to be taken seriously.

So how do we solve this conundrum? Well, first, we can heed [the words of slatestarcodex](#) in the context of traditional politics:

> The rookie mistake is: you see that some system is partly Moloch [ie. captured by misaligned special interests], so you say "Okay, we'll fix that by putting it under the control of this other system. And we'll control this other system by writing 'DO NOT BECOME MOLOCH' on it in bright red marker." ("I see capitalism sometimes gets misaligned. Let's fix it by putting it under control of the government. We'll control the government by having only virtuous people in high offices.") I'm not going to claim there's a great alternative, but the occasionally-adequate alternative is the neoliberal one – find a couple of elegant systems that all optimize along different criteria approximately aligned with human happiness, pit them off against each other in a structure of checks and balances, hope they screw up in different places like in that swiss cheese model, keep enough individual free choice around that people can exit any system that gets too terrible, and let cultural evolution do the rest.

In blockchain governance, it seems like this is the only way forward as well. The approach for blockchain governance that I advocate is "multifactorial consensus", where different coordination flags and different mechanisms and groups are polled, and the ultimate decision depends on the collective result of all of these mechanisms together. These coordination flags may include:

- The roadmap (ie. the set of ideas broadcasted earlier on in the project's history about the direction the project would be going)
- Consensus among the dominant core development teams
- Coin holder votes
- User votes, through some kind of sybil-resistant polling system
- Established norms (eg. non-interference with applications, the 21 million coin limit)

I would argue that it is very useful for coin voting to be one of several coordination institutions

deciding whether or not a given change gets implemented. It is an imperfect and unrepresentative signal, but it is a *Sybil-resistant* one - if you see 10 million ETH voting for a given proposal, you *cannot* dismiss that by simply saying "oh, that's just hired Russian trolls with fake social media accounts". It is also a signal that is sufficiently disjoint from the core development team that if needed it can serve as a check on it. However, as described above, there are very good reasons why it should not be the *only* coordination institution.

And underpinnning it all is the key difference from traditional systems that makes blockchains interesting: the "layer 1" that underpins the whole system is the requirement for individual users to assent to any protocol changes, and their freedom, and credible threat, to "fork off" if someone attempts to force changes on them that they consider hostile (see also: http://vitalik.ca/general/2017/05/08/coordination_problems.html).

Tightly coupled voting is also okay to have in some limited contexts - for example, despite its flaws, miners' ability to vote on the gas limit is a feature that has proven very beneficial on multiple occasions. The risk that miners will try to abuse their power may well be lower than the risk that any specific gas limit or block size limit hard-coded by the protocol on day 1 will end up leading to serious problems, and in that case letting miners vote on the gas limit is a good thing. However, "allowing miners or validators to vote on a few specific parameters that need to be rapidly changed from time to time" is a very far cry from giving them arbitrary control over protocol rules, or letting voting control validation, and these more expansive visions of on-chain governance have a much murkier potential, both in theory and in practice.

A Quick Gasprice Market Analysis

2017 Dec 14

Here is a file that contains data, extracted from geth, about transaction fees in every block between 4710000 and 4730000. For each block, it contains an object of the form:

```
{
    "block":4710000,
    "coinbase":"0x829bd824b016326a401d083b33d092293333a830",
    "deciles":[40,40.1,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,50,66.150044,100]
    ,"free":10248,
    "timedelta":8
}
```

The "deciles" variable contains 11 values, where the lowest is the lowest gasprice in each block, the next is the gasprice that only 10% of other transaction gasprices are lower than, and so forth; the last is the highest gasprice in each block. This gives us a convenient summary of the distribution of transaction fees that each block contains. We can use this data to perform some interesting analyses.

First, a chart of the deciles, taking 50-block moving averages to smooth it out:
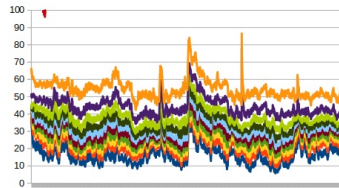
What we see is a gasprice market that seems to actually stay reasonably stable over the course of more than three days. There are a few occasional spikes, most notably the one around block 4720000, but otherwise the deciles all stay within the same band all the way through. The only exception is the highest gasprice transaction (that red squiggle at the top left), which fluctuates wildly because it can be pushed upward by a single very-high-gasprice transaction.

We can try to interpret the data in another way: by calculating, for each gasprice level, the average number of blocks that you need to wait until you see a block where the lowest gasprice included is lower than that gasprice. Assuming that miners are rational and all have the same view (implying that if the lowest gasprice in a block is X, then that means there are no more transactions with gasprices above X waiting to be included), this might be a good proxy for the average amount of time that a transaction sender needs to wait to get included if they use that gasprice. The stats are:
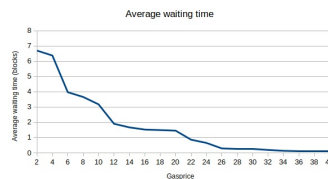
There is clear clustering going on at the 4, 10 and 20 levels; it seems to be an underexploited strategy to send transactions with fees slightly above these levels, getting in before the crowd of transactions right at the level but only paying a little more.

However, there is quite a bit of evidence that miners **do not** have the same view; that is, some miners see a very different set of transactions from other miners. First of all, we can filter blocks by miner address, and check what the deciles of each miner are. Here is the output of this data, splitting by 2000-block ranges so we can spot behavior that is consistent across the entire period, and filtering out miners that mine less than 10 blocks in any period, as well as filtering out blocks with more than 21000 free gas (high levels of free gas may signify an abnormally high minimum gas price policy, like for example 0x6a7a43be33ba930fe58f34e07d0ad6ba7adb9b1f at ~40 gwei and 0xb75d1e62b10e4ba91315c4aa3facc536f8a922f5 at ~10 gwei). We get:

```
0x829bd824b016326a401d083b33d092293333a830 [30, 28, 27, 21, 28, 34, 23, 24, 32, 32]
0xea674fdde714fd979de3edf0f56aa9716b898ec8 [17, 11, 10, 15, 17, 23, 17, 13, 16, 17]
0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c [31, 17, 20, 18, 16, 27, 21, 15, 21, 21]
0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 [20, 16, 19, 14, 17, 18, 17, 14, 15, 15]
0xb2930b35844a230f00e51431acae96fe543a0347 [21, 17, 19, 17, 17, 25, 17, 16, 19, 19]
0x180ba8f73897c0cb26d76265fc7868cfd936e617 [13, 13, 15, 18, 12, 26, 16, 13, 20, 20]
0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb [26, 25, 23, 21, 22, 28, 25, 24, 26, 25]
0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01 [17, 21, 17, 14, 21, 32, 14, 14, 19, 23]
0x2a65aca4d5fc5b5c859090a6c34d164135398226 [26, 24, 20, 16, 22, 33, 20, 18, 24, 24]
```

The first miner is consistently higher than the others; the last is also higher than average, and the second is consistently among the lowest.

Another thing we can look at is timestamp differences - the difference between a block's timestamp and its parent. There is a clear correlation between timestamp difference and lowest gasprice:
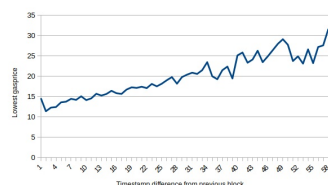
This makes a lot of sense, as a block that comes right after another block should be cleaning up only the transactions that are too low in gasprice for the parent block to have included, and a block that comes a long time after its predecessor would have many more not-yet-included transactions to choose from. The differences are large, suggesting that a single block is enough to bite off a very substantial chunk of the unconfirmed transaction pool, adding to the evidence that most transactions are included quite quickly.

However, if we look at the data in more detail, we see very many instances of blocks with low timestamp differences that contain many transactions with higher gasprices than their parents. Sometimes we do see blocks that actually look like they clean up what their parents could not, like this:

```
{"block":4710093,"coinbase":"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c","deciles":[25,40,40,40,40,40,40,43,50,64.100030001,120],"free":6030,"timedelta":8},
{"block":4710094,"coinbase":"0xea674fdde714fd979de3edf0f56aa9716b898ec8","deciles":[4,16,20,20,21,21,22,29,30,40,59],"free":963366,"timedelta":2},
```

But sometimes we see this:

{"block":4710372,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,30,35,40,40,40,40,40,40,55,100],"free":13320,"timedelta":7},
{"block":4710373,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,32,32,40,40,56,56,56,56,70,80],"free":1672720,"timedelta":2},

And sometimes we see miners suddenly including many 1-gwei transactions:

{"block":4710379,"coinbase":"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c","deciles":[21,25,31,40,40,40,40,40,40,50,80],"free":4979,"timedelta":13},
{"block":4710380,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,1,1,1,1,1,40,45,55,61.10006,2067.909560115],"free":16730,"timedelta":35}

This strongly suggests that a miner including transactions with gasprice X should NOT be taken as evidence that there are not still many transactions with gasprice higher than X left to process. This is likely because of imperfections in network propagation.

In general, however, what we see seems to be a rather well-functioning fee market, though there is still room to improve in fee estimation and, most importantly of all, continuing to work hard to improve base-chain scalability so that more transactions can get included in the first place.

STARKs, Part II: Thank Goodness It's FRI-day

2017 Nov 22

Special thanks to Eli Ben-Sasson for ongoing help and explanations, and Justin Drake for reviewing

In the last part of this series, we talked about how you can make some pretty interesting succinct proofs of computation, such as proving that you have computed the millionth Fibonacci number, using a technique involving polynomial composition and division. However, it rested on one critical ingredient: the ability to prove that at least the great majority of a given large set of points are on the same low-degree polynomial. This problem, called "low-degree testing", is perhaps the single most complex part of the protocol.

We'll start off by once again re-stating the problem. Suppose that you have a set of points, and you claim that they are all on the same polynomial, with degree less than D (ie. $deg < 2$ means they're on the same line, $deg < 3$ means they're on the same line or parabola, etc). You want to create a succinct probabilistic proof that this is actually true.
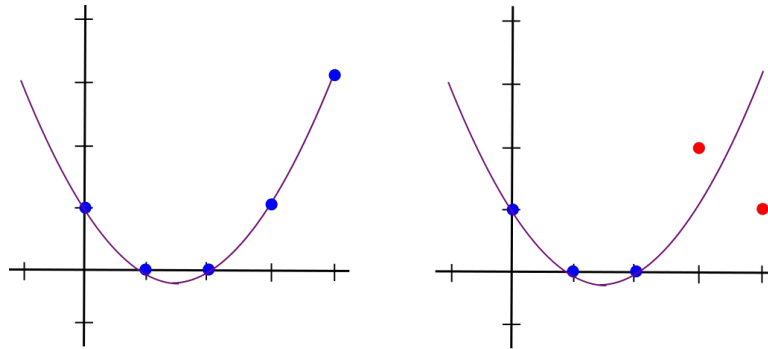
Left: points all on the same $deg < 3$ polynomial. Right: points not on the same $deg < 3$ polynomial

If you want to verify that the points are *all* on the same degree $< D$ polynomial, you would have to actually check every point, as if you fail to check even one point there is always some chance that that point will not be on the polynomial even if all the others are. But what you *can* do is *probabilistically check* that at least *some fraction* (eg. 90%) of all the points are on the same polynomial.
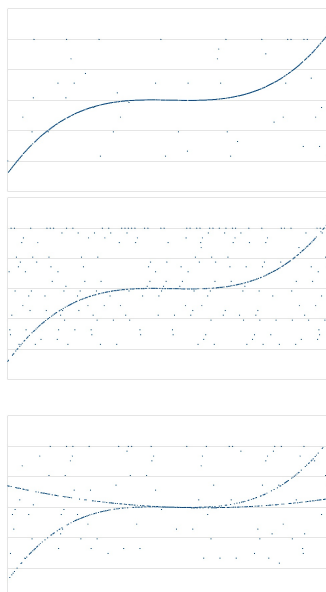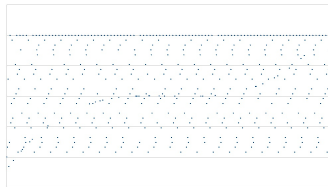
Top left: possibly close enough to a polynomial. Top right: not close enough to a polynomial. Bottom left: somewhat close to two polynomials, but not close enough to either one. Bottom right: definitely not close enough to a polynomial.

If you have the ability to look at *every* point on the polynomial, then the problem is easy. But what if you can only look at a few points - that is, you can ask for whatever specific point you want, and the prover is obligated to give you the data for that point as part of the protocol, but the total number of queries is limited? Then the question becomes, how many points do you need to check to be able to tell with some given degree of certainty?

Clearly, D points is **not** enough. D points are exactly what you need to uniquely define a degree $< D$ polynomial, so *any* set of points that you receive will correspond to *some* degree $< D$ polynomial. As we see in the figure above, however, $D+1$ points or more *do* give some indication.

The algorithm to check if a given set of values is on the same degree $< D$ polynomial with $D+1$ queries is not too complex. First, select a random subset of D points, and use something like Lagrange interpolation (search for "Lagrange interpolation" [here](#) for a more detailed description) to recover the unique degree $< D$ polynomial that passes through all of them. Then, randomly sample one more point, and check that it is on the same polynomial.
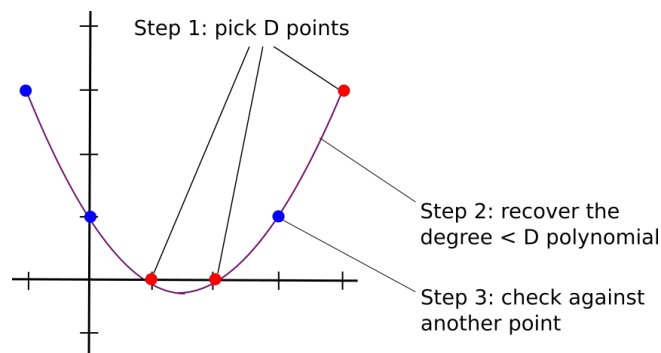
Note that this is only a proximity test, because there's always the possibility that most points are on the same low-degree polynomial, but a few are not, and the $D+1$ sample missed those points entirely. However, we can derive the result that if less than 90% of the points are on the same degree $< D$ polynomial, then the test will fail with high probability. Specifically, if you make $D+k$ queries, and if at least some portion p of the points are not on the same polynomial as the rest of the points, then the test will only pass with probability $(1 - p)^k$.

But what if, as in the examples from the previous article, D is very high, and you want to verify a polynomial's degree with less than D queries? This is, of course, impossible to do directly, because of the simple argument made above (namely, that *any* $k \leq D$ points are all on at least one degree $< D$ polynomial). However, it's quite possible to do this indirectly *by providing auxiliary data*, and achieve massive efficiency gains by doing so. And this is exactly what new protocols like [FRI](#) ("Fast RS IOPP", RS = "[Reed-Solomon](#)", IOPP = "Interactive Oracle Proofs of Proximity"), and similar earlier designs called probabilistically checkable proofs of proximity (PCPPs), try to achieve.
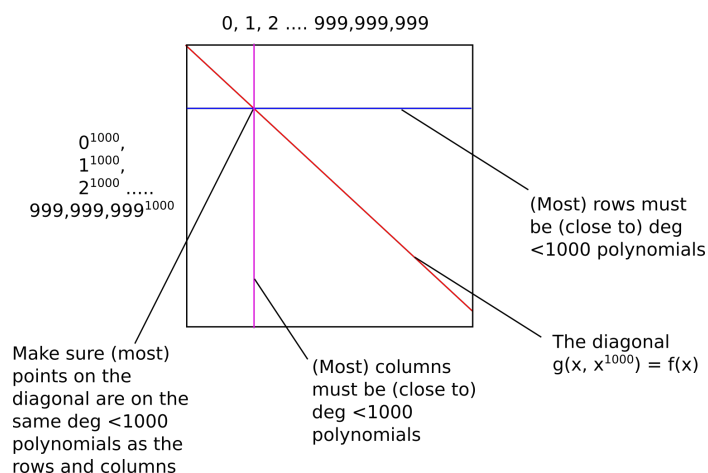
A First Look at Sublinearity

To prove that this is at all possible, we'll start off with a relatively simple protocol, with fairly poor tradeoffs, but that still achieves the goal of sublinear verification complexity - that is, you can prove proximity to a degree $< D$ polynomial with less than D queries (and, for that matter, less than $O(D)$ computation to verify the proof).

The idea is as follows. Suppose there are N points (we'll say N = 1 billion), and they are all on a degree $<$ 1,000,000 polynomial $f(x)$. We find a bivariate polynomial (ie. an expression like $1 + x + xy + x^5 \cdot y^3 + x^{12} + x \cdot y^{11}$), which we will denote $g(x, y)$, such that $g(x, x^{1000}) = f(x)$. This can be done as follows: for the kth degree term in $f(x)$ (eg. $1744 \cdot x^{185423}$), we decompose it into $x^{k \% 1000} \cdot y^{floor(k / 1000)}$ (in this case, $1744 \cdot x^{423} \cdot y^{185}$). You can see that if $y = x^{1000}$, then $1744 \cdot x^{423} \cdot y^{185}$ equals $1744 \cdot x^{185423}$.

In the first stage of the proof, the prover commits to (ie. makes a Merkle tree of) the evaluation of $g(x, y)$ over the *entire* square $[1 ... N] x \{x^{1000}: 1 \leq x \leq N\}$ - that is, all 1 billion x coordinates for the columns, and all 1 billion corresponding *thousandth powers* for the y coordinates of the rows. The diagonal of the square represents the values of $g(x, y)$ that are of the form $g(x, x^{1000})$, and thus correspond to values of $f(x)$.

The verifier then randomly picks perhaps a few dozen rows and columns (possibly using the Merkle root of the square as a source of pseudorandomness if we want a non-interactive proof), and for each row or column that it picks the verifier asks for a sample of, say, 1010 points on the row and column, making sure in each case that one of the points demanded is on the diagonal. The prover must reply back with those points, along with Merkle branches proving that they are part of the original data committed to by the prover. The verifier checks that the Merkle branches match up, and that the points that the prover provides actually do correspond to a degree-1000 polynomial.

0, 1, 2 …. 999,999,999

0^{1000}, 1^{1000}, 2^{1000} ….. $999,999,999^{1000}$

(Most) rows must be (close to) deg <1000 polynomials

Make sure (most) points on the diagonal are on the same deg <1000 polynomials as the rows and columns

(Most) columns must be (close to) deg <1000 polynomials

The diagonal $g(x, x^{1000}) = f(x)$

This gives the verifier a statistical proof that (i) most rows are populated mostly by points on degree < 1000 polynomials, (ii) most columns are populated mostly by points on degree < 1000 polynomials, and (iii) the diagonal line is mostly on these polynomials. This thus convinces the verifier that most points on the diagonal actually do correspond to a degree $< 1,000,000$ polynomial.

If we pick thirty rows and thirty columns, the verifier needs to access a total of 1010 points $\cdot$ 60 rows+cols = 60600 points, less than the original 1,000,000, but not by that much. As far as computation time goes, interpolating the degree < 1000 polynomials will have its own overhead, though since polynomial interpolation can be made subquadratic the algorithm as a whole is still sublinear to verify. The *prover* complexity is higher: the prover needs to calculate and commit to the entire $N \cdot N$ rectangle, which is a total of 10^{18} computational effort (actually a bit more because polynomial evaluation is still superlinear). In all of these algorithms, it will be the case that proving a computation is substantially more complex than just running it; but as we will see the overhead does not have to be *that* high.

A Modular Math Interlude

Before we go into our more complex protocols, we will need to take a bit of a digression into the world of modular arithmetic. Usually, when we work with algebraic expressions and polynomials, we are working with regular numbers, and the arithmetic, using the operators +, -, $\cdot$, / (and exponentiation, which is just repeated multiplication), is done in the usual way that we have all been taught since school: $2 + 2 = 4$, $72 / 5 = 14.4$, $1001 \cdot 1001 = 1002001$, etc. However, what mathematicians have realized is that these ways of defining addition, multiplication, subtraction and division are not the *only* self-consistent ways of defining those operators.

The simplest example of an alternate way to define these operators is modular arithmetic, defined as

follows. The % operator means "take the remainder of": $15 \% 7 = 1$, $53 \% 10 = 3$, etc (note that the answer is always non-negative, so for example $-1 \% 10 = 9$). For any specific prime number p, we can redefine:

$x + y \Rightarrow (x + y) \% p$

$x \cdot y \Rightarrow (x \cdot y) \% p$

$x^y \Rightarrow (x^y) \% p$

$x - y \Rightarrow (x - y) \% p$

$x / y \Rightarrow (x \cdot y^{p-2}) \% p$

The above rules are all self-consistent. For example, if $p = 7$, then:

- $5 + 3 = 1$ (as $8 \% 7 = 1$)
- $1 - 3 = 5$ (as $-2 \% 7 = 5$)
- $2 \cdot 5 = 3$
- $3 / 5 = 2$ (as $(3 \cdot 5^5) \% 7 = 9375 \% 7 = 2$)

More complex identities such as the distributive law also hold: $(2 + 4) \cdot 3$ and $2 \cdot 3 + 4 \cdot 3$ both evaluate to 4. Even formulas like $(a^2 - b^2) = (a - b) \cdot (a + b)$ are still true in this new kind of arithmetic. Division is the hardest part; we can't use regular division because we want the values to always remain integers, and regular division often gives non-integer results (as in the case of $3/5$). The funny $p-2$ exponent in the division formula above is a consequence of getting around this problem using [Fermat's little theorem](#), which states that for any nonzero $x < p$, it holds that $x^{p-1} \% p = 1$. This implies that x^{p-2} gives a number which, if multiplied by x one more time, gives 1, and so we can say that x^{p-2} (which is an integer) equals $\frac{1}{x}$. A somewhat more complicated but faster way to evaluate this modular division operator is the [extended Euclidean algorithm](#), implemented in python [here](#).

Because of how the numbers "wrap around", modular arithmetic is sometimes called "clock math"

With modular math we've created an entirely new system of arithmetic, and because it's self-consistent in all the same ways traditional arithmetic is self-consistent we can talk about all of the same kinds of structures over this field, including polynomials, that we talk about in "regular math". Cryptographers love working in modular math (or, more generally, "finite fields") because there is a bound on the size of a number that can arise as a result of any modular math calculation - no matter what you do, the values will not "escape" the set $\{0, 1, 2 \dots p-1\}$.

Fermat's little theorem also has another interesting consequence. If $p-1$ is a multiple of some number k, then the function $x \rightarrow x^k$ has a small "image" - that is, the function can only give $\frac{p-1}{k} + 1$ possible results. For example, $x \rightarrow x^2$ with $p=17$ has only 9 possible results.
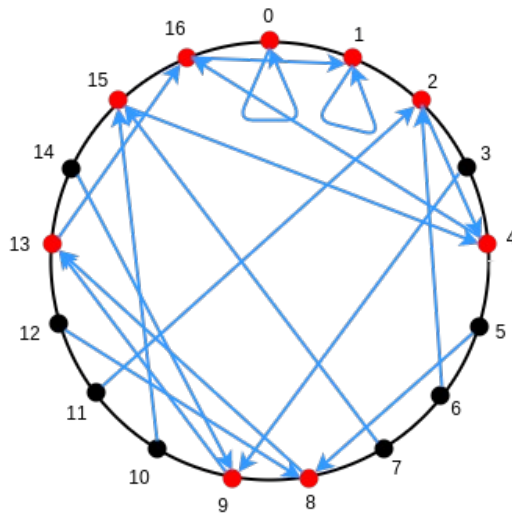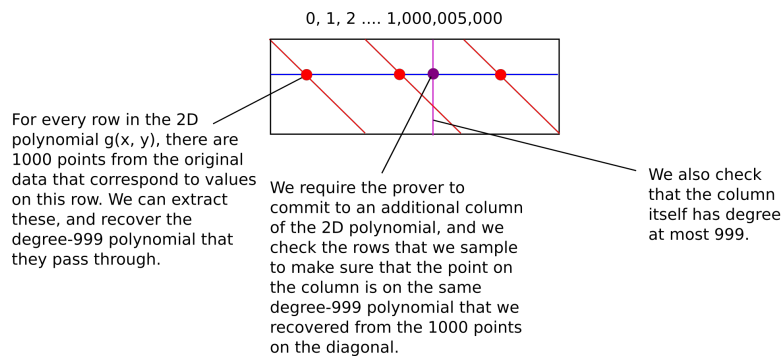
With higher exponents the results are more striking: for example, $x \rightarrow x^8$ with $p=17$ has only 3 possible results. And of course, $x \rightarrow x^{16}$ with $p=17$ has only 2 possible results: for 0 it returns 0, and for everything else it returns 1.
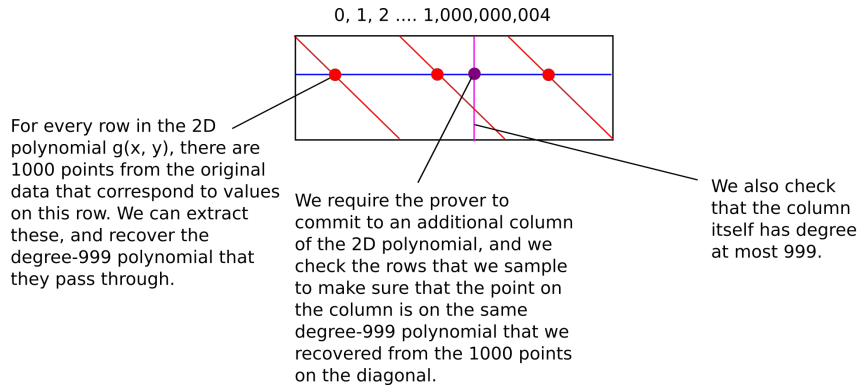
Now A Bit More Efficiency

Let us now move on to a slightly more complicated version of the protocol, which has the modest goal of reducing the prover complexity from 10^{18} to 10^{15}, and then 10^{9}. First, instead of operating over regular numbers, we are going to be checking proximity to polynomials *as evaluated with modular math*. As we saw in the previous article, we need to do this to prevent numbers in our STARKs from growing to 200,000 digits anyway. Here, however, we are going to use the "small image" property of certain modular exponentiations as a side effect to make our protocols far more efficient.

Specifically, we will work with $p =$ 1,000,005,001. We pick this modulus because (i) it's greater than 1 billion, and we need it to be at least 1 billion so we can check 1 billion points, (ii) it's prime, and (iii) $p-1$ is an even multiple of 1000. The exponentiation x^{1000} will have an image of size 1,000,006 - that is, the exponentiation can only give 1,000,006 possible results.

This means that the "diagonal" (x, x^{1000}) now becomes a diagonal with a wraparound; as x^{1000} can only take on 1,000,006 possible values, we only need 1,000,006 rows. And so, the full evaluation of $g(x, x^{1000})$ now has only ~10^{15} elements.

For every row in the 2D polynomial g(x, y), there are 1000 points from the original data that correspond to values on this row. We can extract these, and recover the degree-999 polynomial that they pass through.

We require the prover to commit to an additional column of the 2D polynomial, and we check the rows that we sample to make sure that the point on the column is on the same degree-999 polynomial that we recovered from the 1000 points on the diagonal.

We also check that the column itself has degree at most 999.

As it turns out, we can go further: we can have the prover only commit to the evaluation of g on a single column. The key trick is that the original data itself already contains 1000 points that are on any given row, so we can simply sample those, derive the degree < 1000 polynomial that they are on, and then check that the corresponding point on the column is on the same polynomial. We then check that the column itself is $a < 1000$ polynomial.

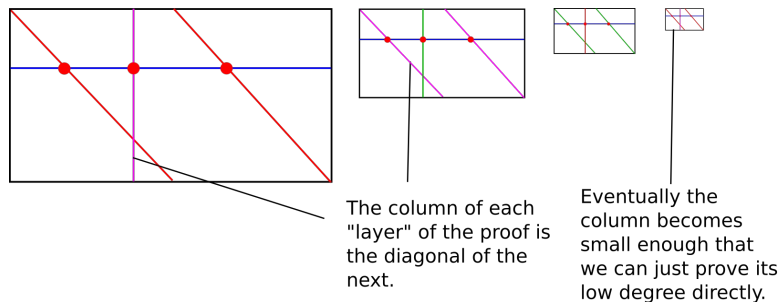0, 1, 2 …. 1,000,000,004

For every row in the 2D polynomial g(x, y), there are 1000 points from the original data that correspond to values on this row. We can extract these, and recover the degree-999 polynomial that they pass through.

We require the prover to commit to an additional column of the 2D polynomial, and we check the rows that we sample to make sure that the point on the column is on the same degree-999 polynomial that we recovered from the 1000 points on the diagonal.

We also check that the column itself has degree at most 999.

The verifier complexity is still sublinear, but the prover complexity has now decreased to 10^9, making it linear in the number of queries (though it's still superlinear in practice because of polynomial evaluation overhead).

And Even More Efficiency

The prover complexity is now basically as low as it can be. But we can still knock the verifier complexity down further, from quadratic to logarithmic. And the way we do *that* is by making the algorithm recursive. We start off with the last protocol above, but instead of trying to embed a polynomial into a 2D polynomial where the degrees in x and y are equal, we embed the polynomial into a 2D polynomial where the degree bound in x is a small constant value; for simplicity, we can even say this must be 2. That is, we express $f(x) = g(x, x^2)$, so that the row check always requires only checking 3 points on each row that we sample (2 from the diagonal plus one from the column).

If the original polynomial has degree $< n$, then the rows have degree < 2 (ie. the rows are straight lines), and the column has degree $< \frac{n}{2}$. Hence, what we now have is a linear-time process for converting a problem of proving proximity to a polynomial of degree $< n$ into a problem of proving proximity to a polynomial of degree $< \frac{n}{2}$. Furthermore, the number of points that need to be committed to, and thus the prover's computational complexity, goes down by a factor of 2 each time (Eli Ben-Sasson likes to compare this aspect of FRI to fast fourier transforms, with the key difference that unlike with FFTs, each step of recursion only introduces one new sub-problem instead of branching out into two). Hence, we can simply keep using the protocol on the column created in the previous round of the protocol, until the column becomes so small that we can simply check it directly; the total complexity is something like $n + \frac{n}{2} + \frac{n}{4} + ... \approx 2n$.

The column of each "layer" of the proof is the diagonal of the next.

Eventually the column becomes small enough that we can just prove its low degree directly.

In reality, the protocol will need to be repeated several times, because there is still a significant probability that an attacker will cheat *one* round of the protocol. However, even still the proofs are not too large; the verification complexity is logarithmic in the degree, though it goes up to $\log^{2}n$ if you count the size of the Merkle proofs.

The "real" FRI protocol also has some other modifications; for example, it uses a binary Galois field (another weird kind of finite field; basically, the same thing as the 12th degree extension fields I talk about here, but with the prime modulus being 2). The exponent used for the row is also typically 4 and not 2. These modifications increase efficiency and make the system friendlier to building STARKs on top of it. However, these modifications are not essential to understanding how the algorithm

works, and if you really wanted to, you could definitely make STARKs with the simple modular math-based FRI described here too.

Soundness

I will warn that *calculating soundness* - that is, determining just how low the probability is that an optimally generated fake proof will pass the test for a given number of checks - is still somewhat of a "here be dragons" area in this space. For the simple test where you take 1,000,000 $(+ k)$ points, there is a simple lower bound: if a given dataset has the property that, for any polynomial, at least portion p of the dataset is not on the polynomial, then a test on that dataset will pass with at most $((1-p)^k)$ probability. However, even that is a very pessimistic lower bound - for example, it's not possible to be much more than 50% close to *two* low-degree polynomials at the same time, and the probability that the first points you select will be the one with the most points on it is quite low. For full-blown FRI, there are also complexities involving various specific kinds of attacks.

Here is a recent article by Ben-Sasson et al describing soundness properties of FRI in the context of the entire STARK scheme. In general, the "good news" is that it seems likely that in order to pass the $(D(x) \cdot Z(x) = C(P(x)))$ check on the STARK, the $(D(x))$ values for an invalid solution would need to be "worst case" in a certain sense - they would need to be maximally far from *any* valid polynomial. This implies that we don't need to check for *that* much proximity. There are proven lower bounds, but these bounds would imply that an actual STARK need to be ~1-3 megabytes in size; conjectured but not proven stronger bounds reduce the required number of checks by a factor of 4.

The third part of this series will deal with the last major part of the challenge in building STARKs: how we actually construct constraint checking polynomials so that we can prove statements about arbitrary computation, and not just a few Fibonacci numbers.

STARKs, Part I: Proofs with Polynomials

2017 Nov 09

Special thanks to Eli Ben-Sasson for ongoing help, explanations and review, coming up with some of the examples used in this post, and most crucially of all inventing a lot of this stuff; thanks to Hsiao-wei Wang for reviewing

Hopefully many people by now have heard of [ZK-SNARKs](), the general-purpose succinct zero knowledge proof technology that can be used for all sorts of usecases ranging from verifiable computation to privacy-preserving cryptocurrency. What you might not know is that ZK-SNARKs have a newer, shinier cousin: ZK-STARKs. With the T standing for "transparent", ZK-STARKs resolve one of the primary weaknesses of ZK-SNARKs, its reliance on a "trusted setup". They also come with much simpler cryptographic assumptions, avoiding the need for elliptic curves, pairings and the knowledge-of-exponent assumption and instead relying purely on hashes and information theory; this also means that they are secure even against attackers with quantum computers.

However, this comes at a cost: the size of a proof goes up from 288 bytes to a few hundred kilobytes. Sometimes the cost will not be worth it, but at other times, particularly in the context of public blockchain applications where the need for trust minimization is high, it may well be. And if elliptic curves break or quantum computers *do* come around, it definitely will be.

So how does this other kind of zero knowledge proof work? First of all, let us review what a general-purpose succinct ZKP does. Suppose that you have a (public) function f, a (private) input x and a (public) output y. You want to prove that you know an x such that $f(x) = y$, without revealing what x is. Furthermore, for the proof to be *succinct*, you want it to be verifiable much more quickly than computing f itself.
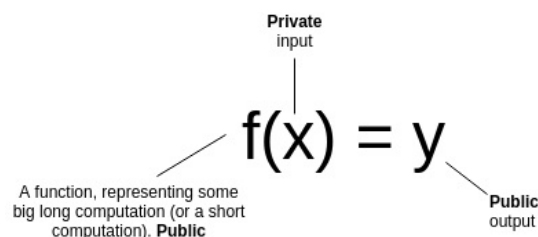

Let's go through a few examples:

- f is a computation that takes two weeks to run on a regular computer, but two hours on a data center. You send the data center the computation (ie. the code to run f), the data center runs it, and gives back the answer y with a proof. You verify the proof in a few milliseconds, and are convinced that y actually is the answer.
- You have an encrypted transaction, of the form "X_1 was my old balance. X_2 was your old balance. X_3 is my new balance. X_4 is your new balance". You want to create a proof that this transaction is valid (specifically, old and new balances are non-negative, and the decrease in my balance cancels out the increase in your balance). x can be the *pair of encryption keys*, and f can be a function which contains as a built-in public input the transaction, takes as input the keys, decrypts the transaction, performs the check, and returns 1 if it passes and 0 if it does not. y would of course be 1.
- You have a blockchain like Ethereum, and you download the most recent block. You want a proof that this block is valid, and that this block is at the tip of a chain where every block in the chain is valid. You ask an existing full node to provide such a proof. x is the entire blockchain (yes, all ?? gigabytes of it), f is a function that processes it block by block, verifies the validity and outputs the hash of the last block, and y is the hash of the block you just downloaded.

So what's so hard about all this? As it turns out, the *zero knowledge* (ie. privacy) guarantee is (relatively!) easy to provide; there are a bunch of ways to convert any computation into an instance of something like the three color graph problem, where a three-coloring of the graph corresponds to a solution of the original problem, and then use a traditional zero knowledge proof protocol to prove that you have a valid graph coloring without revealing what it is. This excellent post by Matthew Green from 2014 describes this in some detail.

The much harder thing to provide is *succinctness*. Intuitively speaking, proving things about computation succinctly is hard because computation is *incredibly fragile*. If you have a long and complex computation, and you as an evil genie have the ability to flip a 0 to a 1 anywhere in the middle of the computation, then in many cases even one flipped bit will be enough to make the computation give a completely different result. Hence, it's hard to see how you can do something like randomly sampling a computation trace in order to gauge its correctness, as it's just too easy to miss that "one evil bit". However, with some fancy math, it turns out that you can.

The general very high level intuition is that the protocols that accomplish this use similar math to what is used in erasure coding, which is frequently used to make *data* fault-tolerant. If you have a piece of data, and you encode the data as a line, then you can pick out four points on the line. Any two of those four points are enough to reconstruct the original line, and therefore also give you the other two points. Furthermore, if you make even the slightest change to the data, then it is guaranteed at least three of those four points. You can also encode the data as a degree-1,000,000 polynomial, and pick out 2,000,000 points on the polynomial; any 1,000,001 of those points will recover the original data and therefore the other points, and any deviation in the original data will change at least 1,000,000 points. The algorithms shown here will make heavy use of polynomials in this way for *error amplification*.
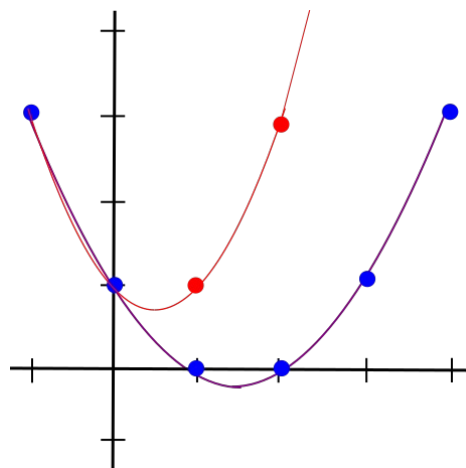

Changing even one point in the original data will lead to large changes in a polynomial's trajectory

A Somewhat Simple Example

Suppose that you want to prove that you have a polynomial P such that $P(x)$ is an integer with $0 \leq P(x) \leq 9$ for all x from 1 to 1 million. This is a simple instance of the fairly common task of "range checking"; you might imagine this kind of check being used to verify, for example, that a set of account balances is still positive after applying some set of transactions. If it were $1 \leq P(x) \leq 9$, this could be part of checking that the values form a correct Sudoku solution.

The "traditional" way to prove this would be to just show all 1,000,000 points, and verify it by checking the values. However, we want to see if we can make a proof that can be verified in less than 1,000,000 steps. Simply randomly checking evaluations of P won't do; there's always the possibility that a malicious prover came up with a P which satisfies the constraint in 999,999 places but does not satisfy it in the last one, and random sampling only a few values will almost always miss that value. So what *can* we do?

Polynomial stays inside
1 <= y <= 9
in this zone

x=1 x=1000000
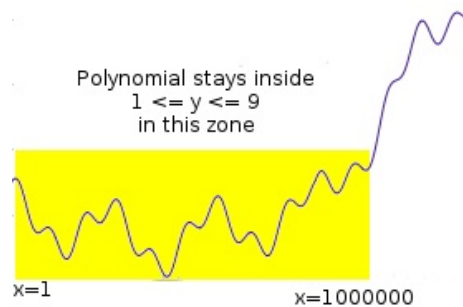
Let's mathematically transform the problem somewhat. Let $C(x)$ be a *constraint checking polynomial*; $C(x) = 0$ if $0 \leq x \leq 9$ and is nonzero otherwise. There's a simple way to construct $C(x)$: $x \cdot (x-1) \cdot (x-2) \cdot \ldots (x-9)$ (we'll assume all of our polynomials and other values use exclusively integers, so we don't need to worry about numbers in between).

40000

20000

2   4   6   8   10

−20000

−40000
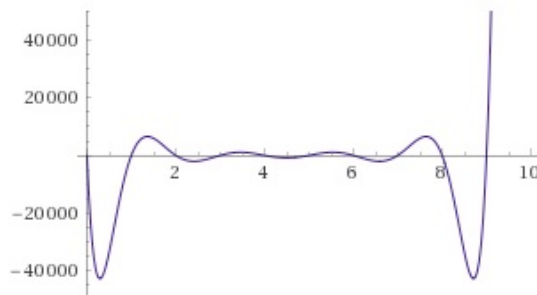
Now, the problem becomes: prove that you know P such that $C(P(x)) = 0$ for all x from 1 to 1,000,000. Let $Z(x) = (x-1) \cdot (x-2) \cdot \ldots (x-1000000)$. It's a known mathematical fact that *any* polynomial which equals zero at all x from 1 to 1,000,000 is a multiple of $Z(x)$. Hence, the problem can now be transformed again: prove that you know P and D such that $C(P(x)) = Z(x) \cdot D(x)$ for all x (note that if you know a suitable $C(P(x))$ then dividing it by $Z(x)$ to compute $D(x)$ is not too difficult; you can use [long polynomial division](#) or more realistically a faster algorithm based on [FFTs](#)). Now, we've converted our original statement into something that looks mathematically clean and possibly quite provable.

So how does one prove this claim? We can imagine the proof process as a three-step communication between a prover and a verifier: the prover sends some information, then the verifier sends some requests, then the prover sends some more information. First, the prover commits to (ie. makes a Merkle tree and sends the verifier the root hash of) the evaluations of $P(x)$ and $D(x)$ for all x from 1 to 1 billion (yes, billion). This includes the 1 million points where $0 \leq P(x) \leq 9$ as well as the 999 million points where that (probably) is not the case.

Values of P(x) here
satisfy
1 <= P(x) <= 9

These values probably
don't, but they're still a very
important part of the proof
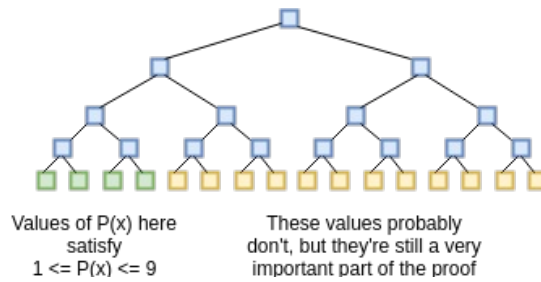
We assume the verifier already knows the evaluation of $Z(x)$ at all of these points; the $Z(x)$ is like a "public verification key" for this scheme that everyone must know ahead of time (clients that do not have the space to store $Z(x)$ in its entirety can simply store the Merkle root of $Z(x)$ and require the prover to also provide branches for every $Z(x)$ value that the verifier needs to query; alternatively, there are some number fields over which $Z(x)$ for certain x is very easy to calculate). After receiving the commitment (ie. Merkle root) the verifier then selects a random 16 x values between 1 and 1 billion, and asks the prover to provide the Merkle branches for $P(x)$ and $D(x)$ there. The prover provides these values, and the verifier checks that (i) the branches match the Merkle root that was provided earlier, and (ii) $C(P(x))$ actually equals $Z(x) \cdot D(x)$ in all 16 cases.

Red and purple: the Merkle
branch proving any
particular value in the
polynomial
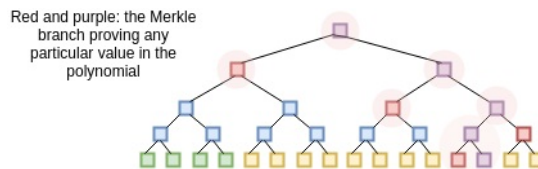
We know that this proof *perfect completeness* - if you actually know a suitable $P(x)$, then if you calculate $D(x)$ and construct the proof correctly it will always pass all 16 checks. But what about *soundness* - that is, if a malicious prover provides a bad $P(x)$, what is the minimum probability that they will get caught? We can analyze as follows. Because $C(P(x))$ is a degree-10 polynomial composed with a degree-1,000,000 polynomial, its degree will be at most 10,000,000. In general, we know that two different degree-N polynomials agree on at most N points; hence, a degree-10,000,000 polynomial which is not equal to any polynomial which always equals $Z(x) \cdot D(x)$ for some x will necessarily disagree with them all at at least 990,000,000 points. Hence, the probability that a bad $P(x)$ will get caught in even one round is already 99%; with 16 checks, the probability of getting caught goes up to $1 - 10^{-32}$; that is to say, the scheme is about as hard to spoof as it is to compute a hash collision.

So... what did we just do? We used polynomials to "boost" the error in any bad solution, so that any incorrect solution to the original problem, which would have required a million checks to find directly, turns into a solution to the verification protocol that can get flagged as erroneous at 99% of the time with even a single check.

We can convert this three-step mechanism into a *non-interactive proof*, which can be broadcasted by a single prover once and then verified by anyone, using the [Fiat-Shamir heuristic](). The prover first builds up a Merkle tree of the $P(x)$ and $D(x)$ values, and computes the root hash of the tree. The root itself is then used as the source of entropy that determines what branches of the tree the prover needs to provide. The prover then broadcasts the Merkle root and the branches together as the proof. The computation is all done on the prover side; the process of computing the Merkle root from the data, and then using that to select the branches that get audited, effectively substitutes the need for an interactive verifier.

The only thing a malicious prover without a valid $P(x)$ can do is try to make a valid proof over and over again until eventually they get *extremely* lucky with the branches that a Merkle root that they compute selects, but with a soundness of $1 - 10^{-32}$ (ie. probability of at least $1 - 10^{-32}$ that a given attempted fake proof will fail the check) it would take a malicious prover billions of years to make a passable proof.

Step 2: Calculate Merkle root hash

Step 1: Gather underlying data

Step 3: Use root hash as pseudorandom seed to randomly select Merkle branches to audit
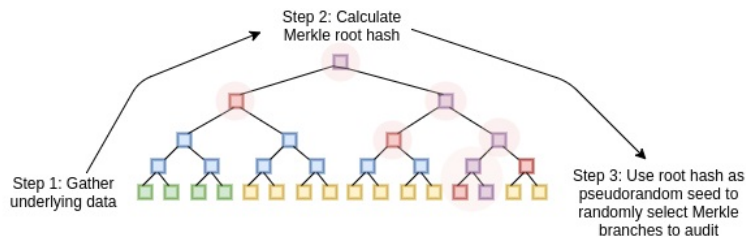
Going Further

To illustrate the power of this technique, let's use it to do something a little less trivial: prove that you know the millionth Fibonacci number. To accomplish this, we'll prove that you have knowledge of a polynomial which represents a computation tape, with $P(x)$ representing the xth Fibonacci number. The constraint checking polynomial will now hop across three x-coordinates: $C(x_1, x_2, x_3) = x_3 - x_2 - x_1$ (notice how if $C(P(x), P(x+1), P(x+2)) = 0$ for all x then $P(x)$ represents a Fibonacci sequence).

$$C(P(4), P(5), P(6))$$
$$= P(6) - P(5) - P(4)$$
$$= 0$$
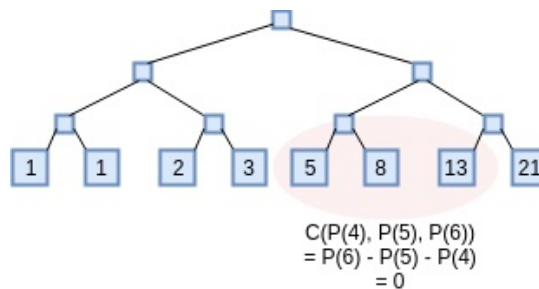
The translated problem becomes: prove that you know P and D such that $C(P(x), P(x+1), P(x+2)) = Z(x) \cdot D(x)$. For each of the 16 indices that the proof audits, the prover will need to provide Merkle branches for $P(x)$, $P(x+1)$, $P(x+2)$ and $D(x)$. The prover will additionally need to provide Merkle branches to show that $P(0) = P(1) = 1$. Otherwise, the entire process is the same.

Now, to accomplish this in reality there are two problems that need to be resolved. The first problem is that if we actually try to work with regular numbers the solution would not be efficient *in practice*, because the numbers themselves very easily get extremely large. The millionth Fibonacci number, for example, has 208988 digits. If we actually want to achieve succinctness in practice, instead of doing these polynomials with regular numbers, we need to use finite fields - number systems that still follow the same arithmetic laws we know and love, like $a \cdot (b+c) = (a\cdot b) + (a\cdot c)$ and $(a^2 - b^2) = (a-b) \cdot (a+b)$, but where each number is guaranteed to take up a constant amount of space. Proving claims about the millionth Fibonacci number would then require a more complicated design that implements big-number arithmetic *on top of* this finite field math.

The simplest possible finite field is modular arithmetic; that is, replace every instance of $a + b$ with $a + b \mod{N}$ for some prime N, do the same for subtraction and multiplication, and for division use [modular inverses](#) (eg. if $N = 7$, then $3 + 4 = 0$, $2 + 6 = 1$, $3 \cdot 4 = 5$, $4 / 2 = 2$ and $5 / 2 = 6$). You can learn more about these kinds of number systems from my description on prime fields [here](#) (search "prime field" in the page) or this [Wikipedia article](#) on modular arithmetic (the articles that you'll find by searching directly for "finite fields" and "prime fields" unfortunately tend to be very complicated and go straight into abstract algebra, don't bother with those).

Second, you might have noticed that in my above proof sketch for soundness I neglected to cover one kind of attack: what if, instead of a plausible degree-1,000,000 $P(x)$ and degree-9,000,000 $D(x)$, the attacker commits to some values that are not on *any* such relatively-low-degree polynomial? Then, the argument that an invalid $C(P(x))$ must differ from any valid $C(P(x))$ on at least 990 million points does not apply, and so different and much more effective kinds of attacks *are* possible. For example, an attacker could generate a random value p for every x, then compute $d = C(p) / Z(x)$ and commit to these values in place of $P(x)$ and $D(x)$. These values would not be on any kind of low-degree polynomial, but they *would* pass the test.

It turns out that this possibility can be effectively defended against, though the tools for doing so are fairly complex, and so you can quite legitimately say that they make up the bulk of the mathematical

innovation in STARKs. Also, the solution has a limitation: you can weed out commitments to data that are *very* far from any degree-1,000,000 polynomial (eg. you would need to change 20% of all the values to make it a degree-1,000,000 polynomial), but you cannot weed out commitments to data that only differ from a polynomial in only one or two coordinates. Hence, what these tools will provide is *proof of proximity* - proof that *most* of the points on P and D correspond to the right kind of polynomial.

As it turns out, this is sufficient to make a proof, though there are two "catches". First, the verifier needs to check a few more indices to make up for the additional room for error that this limitation introduces. Second, if we are doing "boundary constraint checking" (eg. verifying $P(0) = P(1) = 1$) in the Fibonacci example above), then we need to extend the proof of proximity to not only prove that most points are on the same polynomial, but also prove that *those two specific points* (or whatever other number of specific points you want to check) are on that polynomial.

In the next part of this series, I will describe the solution to proximity checking in much more detail, and in the third part I will describe how more complex constraint functions can be constructed to check not just Fibonacci numbers and ranges, but also arbitrary computation.

On Medium-of-Exchange Token Valuations

2017 Oct 17

One kind of token model that has become popular among many recent token sale projects is the "network medium of exchange token". The general pitch for this kind of token goes as follows. We, the developers, build a network, and this network allows you to do new cool stuff. This network is a sharing-economy-style system: it consists purely of a set of sellers, that provide resources within some protocol, and buyers that purchase the services, where both buyers and sellers come from the community. But the purchase and sale of things within this network *must* be done with the new token that we're selling, and this is why the token will have value.

If it were the developers themselves that were acting as the seller, then this would be a very reasonable and normal arrangement, very similar in nature to a Kickstarter-style product sale. The token actually would, in a meaningful economic sense, be backed by the services that are provided by the developers.

We can see this in more detail by describing what is going on in a simple economic model. Suppose that N people value a product that a developer wants to release at $\$x$, and believe the developer will give them the product. The developer does a sale, and raises N units for $\$w < x$ each, thus raising a total revenue of $\$Nw$. The developer builds the product, and gives it to each of the buyers. At the end of the day, the buyers are happy, and the developer is happy. Nobody feels like they made an avoidable mistake in participating, and everyone's expectations have been met. This kind of economic model is clearly stable.

Now, let's look at the story with a "medium of exchange" token. N people value a product that will exist in a decentralized network at $\$x$; the product will be sold at a price of $\$w < x$. They each buy $\$w$ of tokens in the sale. The developer builds the network. Some sellers come in, and offer the product inside the network for $\$w$. The buyers use their tokens to purchase this product, spending $\$w$ of tokens and getting $\$x$ of value. The sellers spend $\$v < w$ of resources and effort producing this product, and they now have $\$w$ worth of tokens.

Notice that here, the cycle is not complete, and in fact it never will be; there needs to be an ongoing stream of buyers and sellers for the token to continue having its value. The stream does not strictly speaking *have to be* endless; if in every round there is a chance of at least $\frac{v}{w}$ that there will be a next round, then the model still works, as even though *someone* will eventually be cheated, the risk of any individual participant becoming that person is lower than the benefit that they get from participating. It's also totally possible that the token would depreciate in each round, with its value multiplying by some factor f where $\frac{v}{w} < f < 1$, until it eventually reaches a price of zero, and it would still be on net in everyone's interest to participate. Hence, the model is theoretically feasible, but you can see how this model is more complex and more tenuous than the simple "developers as seller" model.

Traditional macroeconomics has a [simple equation](#) to try to value a medium of exchange:

$MV = PT$

Here:

- M is the total money supply; that is, the total number of coins
- V is the "velocity of money"; that is, the number of times that an average coin changes hands every day
- P is the "price level". This is the price of goods and services *in terms of* the token; so it is actually the *inverse* of the currency's price
- T is the transaction volume: the economic value of transactions per day

The proof for this is a trivial equality: if there are N coins, and each changes hands M times per day, then this is $M \cdot N$ coins' worth of economic value transacted per day. If this represents $\$T$ worth of economic value, then the price of each coin is $\frac{T}{M \cdot N}$, so the "price

level" is the inverse of this, $\frac{M \cdot N}{T}$.

For easier analysis, we can recast two variables:

- We refer to $\frac{1}{V}$ with H, the time that a user holds a coin before using it to make a transaction
- We refer to $\frac{1}{P}$ with C, the price of the currency (think $C = cost$)

Now, we have:

$\frac{M}{H} = \frac{T}{C}$

$MC = TH$

The left term is quite simply the market cap. The right term is the economic value transacted per day, multiplied by the amount of time that a user holds a coin before using it to transact.

This is a steady-state model, assuming that the same quantity of users will also be there. In reality, however, the quantity of users may change, and so the price may change. The time that users hold a coin may change, and this may cause the price to change as well.

Let us now look once again at the economic effect on the users. *What do users lose* by using an application with a built-in appcoin rather than plain old ether (or bitcoin, or USD)? The simplest way to express this is as follows: the "implicit cost" imposed by such a system on users *the cost to the user of holding those coins for that period of time, instead of holding that value in the currency that they would otherwise have preferred to hold*.

There are many factors involved in this cost: cognitive costs, exchange costs and spreads, transaction fees, and many smaller items. One particular significant factor of this implicit cost is expected return. If a user expects the appcoin to only grow in value by 1% per year, while their other available alternatives grow 3% per year, and they hold $20 of the currency for five days, then that is an expected loss of roughly $\$20 \cdot 2\% \cdot 5 / 365 = \$0.0054$.

One immediate conclusion from this particular insight is that **appcoins are very much a multi-equilibrium game**. If the appcoin grows at 2% per year, then the fee drops to $0.0027, and this essentially makes the "de-facto fee" of the application (or at least a large component of it) 2x cheaper, attracting more users and growing its value more. If the appcoin starts falling at 10% per year, however, then the "de-facto fee" grows to $0.035, driving many users away and accelerating its growth.

This leads to increased opportunities for market manipulation, as a manipulator would not just be wasting their money fighting against a single equilibrium, but may in fact successfully nudge a given currency from one equilibrium into another, and profit from successfully "predicting" (ie. causing) this shift. It also means there is a large amount of path dependency, and established brands matter a lot; witness the epic battles over which fork of the bitcoin blockchain can be called Bitcoin for one particular high-profile example.

Another, and perhaps even more important, conclusion is that the market cap of an appcoin **depends crucially on the holding time H**. If someone creates a very efficient exchange, which allows users to purchase an appcoin in real time and then immediately use it in the application, then allowing sellers to immediately cash out, then the market cap would drop precipitously. If a currency is stable or prospects are looking optimistic, then this may not matter because users actually see no disadvantage from holding the token instead of holding something else (ie. zero "de-facto fee"), but if prospects start to turn sour then such a well-functioning exchange can acelerate its demise.

You might think that exchanges are inherently inefficient, requiring users to create an account, login, deposit coins, wait for 36 confirmations, trade and logout, but in fact hyper-efficient exchanges are around the corner. [Here](#) is a thread discussing designs for fully autonomous synchronous on-chain transactions, which can convert token A into token B, and possibly even then use token B to do something, *within a single transaction*. Many other platforms are being developed as well.

What this all serves to show is that relying purely on the medium-of-exchange argument to support a token value, while attractive because of its seeming ability to print money out of thin air, is ultimately quite brittle. Protocol tokens using this model may well be sustained for some time due to irrationality and temporary equilibria where the implicit cost of holding the token is zero, but it is a kind of model which always has an unavoidable risk of collapsing at any time.

So what is the alternative? One simple alternative is the etherdelta approach, where an application simply collects fees in the interface. One common criticism is: but can't someone fork the interface to take out the fees? A counter-retort is: someone can also fork the interface to replace your protocol token with ETH, BTC, DOGE or whatever else users would prefer to use. One can make a more sophisticated argument that this is hard because the "pirate" version would have to compete with the "official" version for network effect, but one can just as easily create an official fee-paying client that refuses to interact with non-fee-paying clients as well; this kind of network effect-based enforcement is similar to how value-added-taxes are typically enforced in Europe and other places. Official-client buyers would not interact with non-official-client sellers, and official-client sellers would not interact with non-official-client buyers, so a large group of users would need to switch to the "pirate" client at the same time to successfully dodge fees. This is not perfectly robust, but it is certainly as good as the approach of creating a new protocol token.

If developers want to front-load revenue to fund initial development, then they can sell a token, with the property that all fees paid are used to buy back some of the token and burn it; this would make the token backed by the future expected value of upcoming fees spent inside the system. One can transform this design into a more direct utility token by requiring users to use the utility token to pay fees, and having the interface use an exchange to automatically purchase tokens if the user does not have tokens already.

The important thing is that for the token to have a stable value, it is highly beneficial for the token supply to have **sinks** - places where tokens actually disappear and so the total token quantity decreases over time. This way, there is a more transparent and explicit fee paid by users, instead of the highly variable and difficult to calculate "de-facto fee", and there is also a more transparent and explicit way to figure out what the value of protocol tokens should be.

A Prehistory of the Ethereum Protocol

2017 Sep 14

Although the ideas behind the current Ethereum protocol have largely been stable for two years, Ethereum did not emerge all at once, in its current conception and fully formed. Before the blockchain has launched, the protocol went through a number of significant evolutions and design decisions. The purpose of this article will be to go through the various evolutions that the protocol went through from start to launch; the countless work that was done on the implementations of the protocol such as Geth, cppethereum, pyethereum, and EthereumJ, as well as the history of applications and businesses in the Ethereum ecosystem, is deliberately out of scope.

Also out of scope is the history of Casper and sharding research. While we can certainly make more blog posts talking about all of the various ideas Vlad, Gavin, myself and others came up with, and discarded, including "proof of proof of work", hub-and-spoke chains, "hypercubes", shadow chains (arguably a precursor to Plasma), chain fibers, and various iterations of Casper, as well as Vlad's rapidly evolving thoughts on reasoning about incentives of actors in consensus protocols and properties thereof, this would also be far too complex a story to go through in one post, so we will leave it out for now.

Let us first begin with the very earliest version of what would eventually become Ethereum, back when it was not even called Ethereum. When I was visiting Israel in October 2013, I spent quite a bit of time with the Mastercoin team, and even suggested a few features for them. After spending a couple of times thinking about what they were doing, I sent the team a proposal to make their protocol more generalized and support more types of contracts without adding an equally large and complex set of features:

https://web.archive.org/web/20150627031414/http://vbuterin.com/ultimatescripting.html

Ultimate Scripting: A Platform for Generalized Financial Contracts on Mastercoin

0.1. Introduction

Perhaps the key advantage of Mastercoin over the raw Bitcoin protocol is the potential to include much more advanced transaction types, including transactions that specify behavior based on future information well off into the future. For example, Mastercoin joins Ripple in being one of the only two major cryptocurrency networks that include the ability for users to make binding exchange offers as a type of transaction. From there, the Mastercoin Foundation intends to integrate even more complex contracts, including bets, contracts for difference and on-blockchain dice rolls. However, up until this point Mastercoin has been taking a relatively unstructured process in developing these idea, essentially treating each one as a separate "feature" with its own transaction code and rules. This document outlines an alternative way of specifying Mastercoin contracts which follows an open-ended philosophy, specifying only the basic data and arithmetic building blocks and allowing anyone to craft arbitrarily complex Mastercoin contracts to suit their own needs, including needs which we may not even anticipate.

0.2. Specification

The underlying idea behind this specification is to allow anyone to create a contract which pays out according to an arbitrary formula. The formula will be defined in a Bitcoin-like stack-based scripting language, consisting of numbers and opcodes.

The evaluation algorithm is as follows:

```
dataStack = []
opStack = script
while len(opStack) > 0:
    var op = opStack.pop()
    if typeof(op) == 'opcode': eval(dataStack,op)
    else: dataStack.push(op)
return dataStack.pop()
```

Where `eval` is defined for each opcode below. Any error (eg. division by zero) will make the script return `FAIL`, and result in the entire transaction being treated as invalid by the Mastercoin network. All variables will be signed 64-bit integers, and all arithmetic operations wrap around (that is, if the underlying arithmetic operation returns R, the value pushed is $((R + 2^{63}) \% 2^{64}) - 2^{63}$.

Notice that this is very far from the later and more expansive vision of Ethereum: it specialized purely in what Mastercoin was trying to specialize in already, namely two-party contracts where parties A and B would both put in money, and then they would later get money out according to some formula specified in the contract (eg. a bet would say "if X happens then give all the money to A, otherwise give all the money to B"). The scripting language was not Turing-complete.

The Mastercoin team was impressed, but they were not interested in dropping everything they were doing to go in this direction, which I was increasingly convinced is the correct choice. So here comes version 2, circa December:

https://web.archive.org/web/20131219030753/http://vitalik.ca/ethereum.html

Ethereum: The Ultimate Smart Contract and Autonomous Corporation Platform on the Blockchain

In the last few months, there has been a great amount of interest into the area of using the Bitcoin blockchain, the mechanism that allows for the entire world to agree on the state of a public ownership database, for more than just money. Perhaps the first, and oldest, such alternative application is colored coins, which is a protocol that allows users to label specific bitcoins and treat them as assets representing some real world value - whether company shares, collectibles or even existing currencies like gold and USD. A more independent alternative, Ripple, also includes the ability to create custom currencies and assets, but adds a decentralized exchange. More recently, Mastercoin has started to go even further, allowing more complex financial contracts such as hedging, trust-free dice rolls, binary options and self-stabilizing currencies - essentially, almost any common financial instrument imaginable. Taken together, all of these projects can be thought of as initial efforts toward a sort of "cryptocurrency 2.0" - they are to Bitcoin what Web 2.0 was to the World Wide Web circa 1995.

At the same time, there has been significant interest in "decentralized autonomous corporations" - autonomous entities that operate on the blockchain in a completely transparent and publicly managed way without any central control whatsoever. Rather than the relationships of the investors, owners and employees of the corporation being mediated by a legal contract or a set of organizational bylaws, the funds and corporate resources are managed directly on the blockchain. However, decentralized autonomous corporations are difficult to implement today, simply because the scripting systems of Bitcoin, and even proto-cryptocurrency 2.0 alternatives like Ripple and Mastercoin, are far too limited to allow the kind of arbitrarily complex computation that DACs require. Although these platforms have begun to offer increasingly complex contracts such as financial derivatives, order matching and trust-free bets, the way that the protocols are set up is inherently limited and closed-ended: each of these use cases is treated as a specific transaction type, not allowing any way for users to build contracts that the developers have not specifically chosen to include.

What this project intends to do is take cryptocurrency 2.0, and generalize it - create a fully-fledged, Turing-complete (but heavily fee-regulated) cryptographic ledger that allows participants to encode arbitrarily complex contracts, autonomous agents and relationships that will be mediated entirely by the blockchain. On-chain currencies, futures contracts, prediction markets, Namecoin-style domain name systems and even provably fair gambling sites will become trivial to implement, existing as simple, hundred-line-of-code contracts on the chain.

Basic Building Blocks

Here you can see the results of a substantial rearchitecting, largely a result of a long walk through San Francisco I took in November once I realized that smart contracts could potentially be fully generalized. Instead of the scripting language being simply a way of describing the terms of relations between two parties, contracts were themselves fully-fledged accounts, and had the ability to hold, send and receive assets, and even maintain a permanent storage (back then, the permanent storage was called "memory", and the only temporary "memory" was the 256 registers). The language switched from being a stack-based machine to being a register-based one on my own volition; I had little argument for this other than that it seemed more sophisticated.

Additionally, notice that there is now a built-in fee mechanism:

Whenever ether is sent to a script, the following happens:

1. The ether's endowment increases by the amount sent
2. All registers are reset to zero.
3. The sender is placed into R0.
4. The value sent is placed into R1.
5. The fee is placed into R2.
6. The index pointer is set to zero, and `STEPCOUNT = 0`
7. Repeat forever:
   - set `TOTALFEE = 0`
   - set `STEPCOUNT <- STEPCOUNT + 1`
   - if `STEPCOUNT > 16`, set `TOTALFEE <- TOTALFEE + STEPFEE`
   - see if the command at the index pointer is a valid command and not `STOP`. If it is invalid or `STOP`, `HALT` and break out of the loop
   - see if the command will do any modifications to the contract. If so, set `TOTALFEE <- TOTALFEE + DATAFEE`
   - see if the command will fill up a previously zero memory field. If so, set `TOTALFEE <- TOTALFEE + MEMORYFEE`
   - see if the commend will zero a previously used memory field. If so, set `TOTALFEE <- TOTALFEE - MEMORYFEE`
   - see if the command is `EXTRO` or `BALANCE`. If so, set `TOTALFEE <- TOTALFEE + EXTROFEE`
   - see if the command is `MKTX` or `RAWTX`. If so, set `TOTALFEE <- TOTALFEE + (transaction's value plus transaction's fee)`
   - if `TOTALFEE > contract's endowment`, `HALT` and break out of the loop
   - else, subtract `TOTALFEE` from contract's endowment. Note that `TOTALFEE` may be negative in some cases, in which case the endowment would actually increase
   - run the command

At this point, ether literally was gas; after every single computational step, the balance of the contract that a transaction was calling would drop a little bit, and if the contract ran out of money execution would halt. Note that this "receiver pays" mechanism meant that the contract itself had to require the sender to pay the contract a fee, and immediately exit if this fee is not present; the protocol allocated an allowance of 16 free execution steps to allow contracts to reject non-fee-paying transactions.

This was the time when the Ethereum protocol was entirely my own creation. From here on, however, new participants started to join the fold. By far the most prominent on the protocol side was Gavin Wood, who reached out to me in an about.me message in December 2013:

Jeffrey Wilcke, lead developer of the Go client (back then called "ethereal") also reached out and started coding around the same time, though his contributions were much more on the side of client development rather than protocol research.

*"Hey Jeremy, glad to see you're interested in Ethereum..."*

Gavin's initial contributions were two-fold. First, you might notice that the contract calling model in the initial design was an asynchronous one: although contract

A could create an "internal transaction" to contract B ("internal transaction" is Etherscan's lingo; initially they were just called "transactions" and then later "message calls" or "calls"), the internal transaction's execution would not start until the execution of the first transaction completely finished. This meant that transactions could not use internal transactions as a way of getting information from other contracts; the only way to do that was the EXTRO opcode (kind of like an SLOAD that you could use to read other contracts' storage), and this too was later removed with the support of Gavin and others.

When implementing my initial spec, Gavin naturally implemented internal transactions synchronously without even realizing that the intent was different - that is to say, in Gavin's implementation, when a contract calls another contract, the internal transaction gets executed immediately, and once that execution finishes, the VM returns back to the contract that created the internal transaction and proceeds to the next opcode. This approach seemed to both of us to be superior, so we decided to make it part of the spec.

Second, a discussion between him and myself (during a walk in San Francisco, so the exact details will be forever lost to the winds of history and possibly a copy or two in the deep archives of the NSA) led to a re-factoring of the transaction fee model, moving away from the "contract pays" approach to a "sender pays" approach, and also switching to the "gas" architecture. Instead of each individual transaction step immediately taking away a bit of ether, the transaction sender pays for and is allocated some "gas" (roughly, a counter of computational steps), and computational steps drew from this allowance of gas. If a transaction runs out of gas, the gas would still be forfeit, but the entire execution would be reverted; this seemed like the safest thing to do, as it removed an entire class of "partial execution" attacks that contracts previously had to worry about. When a transaction execution finishes, the fee for any unused gas is refunded.

Gavin can also be largely credited for the subtle change in vision from viewing Ethereum as a platform for building programmable money, with blockchain-based contracts that can hold digital assets and transfer them according to pre-set rules, to a general-purpose computing platform. This started with subtle changes in emphasis and terminology, and later this influence became stronger with the increasing emphasis on the "Web 3" ensemble, which saw Ethereum as being one piece of a suite of decentralized technologies, the other two being Whisper and Swarm.
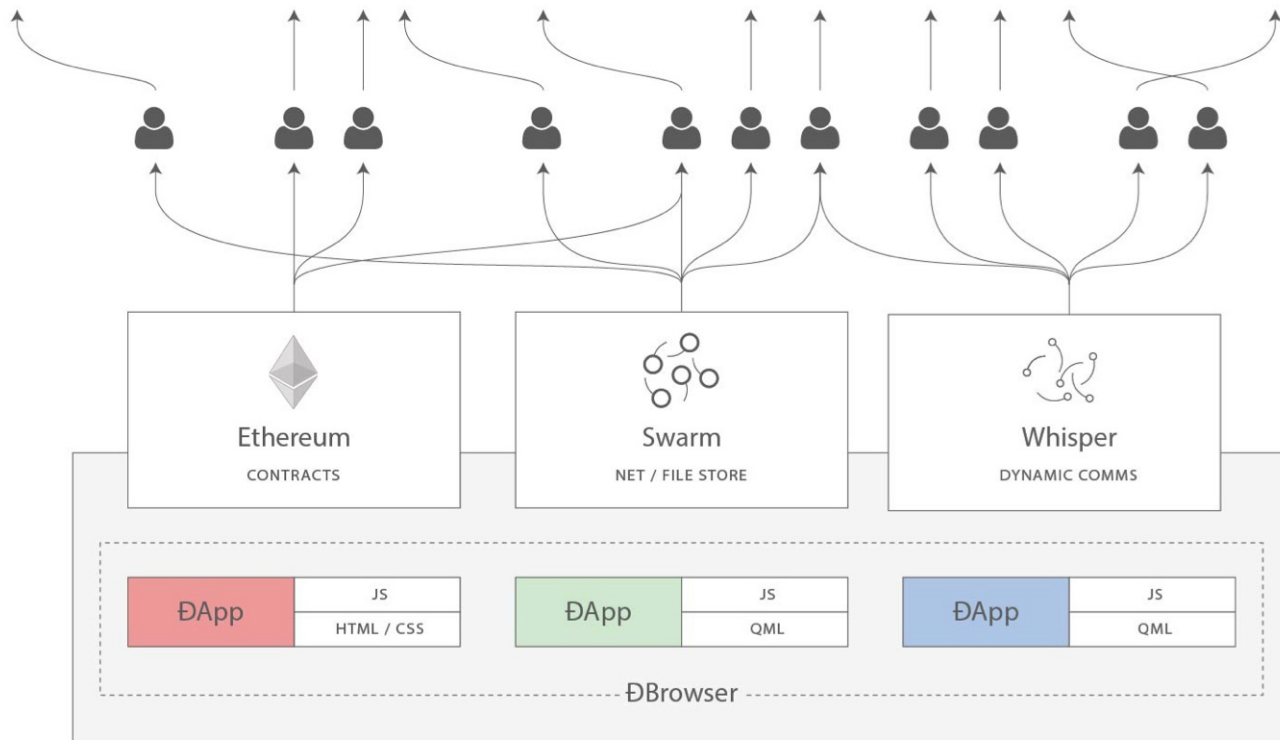
There were also changes made around the start of 2014 that were suggested by others. We ended up moving back to a stack-based architecture after the idea was suggested by Andrew Miller and others.

Charles Hoskinson suggested the switch from Bitcoin's SHA256 to the newer SHA3 (or, more accurately, keccak256). Although there was some controversy for a while, discussions with Gavin, Andrew and others led to establishing that the size of values on the stack should be limited to 32 bytes; the other alternative being considered, unlimited-size integers, had the problem that it was too difficult to figure out how much gas to charge for additions, multiplications and other operations.

The initial mining algorithm that we had in mind, back in January 2014, was a contraption called Dagger:

https://github.com/ethereum/wiki/blob/master/Dagger.md

Algorithm specification:

Essentially, the Dagger algorithm works by creating a directed acyclic graph (the technical term for a tree where each node is allowed to have multiple parents) with a total of 2^{23} - 1 nodes in sequence. Each node depends on 3-15 randomly selected nodes before it. If the miner finds a node between index 2^{22} and 2^{23} such that this resulting hash is below 2^{256} divided by the difficulty parameter, the result is a valid proof of work.

Let `D` be the underlying data (eg. in Bitcoin's case the block header), `N` be the nonce and `||` be the string concatenation operator (ie. `'foo' || 'bar' == 'foobar'`). The entire code for the algorithm is as follows:

```
D(data,xn,0) = sha3(data)
D(data,xn,n) =
    with v = sha3(data + xn + n)
        L = 2 if n < 2^21 else 11 if n < 2^22 else 3
        a[k] = floor(v/n^k) mod n for 0 <= k < 2
        a[k] = floor(v/n^k) mod 2^22 for 2 <= k < L
    sha3(v ++ D(data,xn,a[0]) ++ D(data,xn,a[1]) ++ ... ++ D(data,xn,a[L-1]))
```

Properties:

Objective: find `xn` , `n` such that `n > 2^22` and `D(data,xn,n) < 2^256 / diff`

Dagger was named after the "directed acyclic graph" (DAG), the mathematical structure that is used in the algorithm. The idea is that every N blocks, a new DAG would be pseudorandomly generated from a seed, and the bottom layer of the DAG would be a collection of nodes that takes several gigabytes to store. However, generating any individual value in the DAG would require calculating only a few thousand entries. A "Dagger computation" involved getting some number of values in random positions in this bottom-level dataset and hashing them together. This meant that there was a fast way to make a Dagger calculation - already having the data in memory, and a slow, but not memory intensive way - regenerating each value from the DAG that you need to get from scratch.

The intention of this algorithm was to have the same "memory-hardness" properties as algorithms that were popular at the time, like Scrypt, but still be light-client friendly. Miners would use the fast way, and so their mining would be constrained by memory bandwidth (the theory is that consumer-grade RAM is already very heavily optimized, and so it would be hard to further optimize it with ASICs), but light clients could use the memory-free but slower version for verification. The fast way might take a few microseconds and the slow but memory-free way a few milliseconds, so it would still be very viable for light clients.

From here, the algorithm would change several times over the course of Ethereum development. The next idea that we went through is "adaptive proof of work"; here, the proof of work would involve executing randomly selected Ethereum contracts, and there is a clever reason why this is expected to be ASIC-resistant: if an ASIC was developed, competing miners would have the incentive to create and publish many contracts that that ASIC was not good at executing. There is no such thing as an ASIC for general computation, the story goes, as that is just a CPU, so we could instead use this kind of adversarial incentive mechanism to make a proof of work that essentially was executing general computation.

This fell apart for one simple reason: long-range attacks. An attacker could start a chain from block 1, fill it up with only simple contracts that they can create specialized hardware for, and rapidly overtake the main chain. So... back to the drawing board.

The next algorithm was something called Random Circuit, described in this google doc here, proposed by myself and Vlad Zamfir, and analyzed by Matthew Wampler-Doty and others. The idea here was also to simulate general-purpose computation inside a mining algorithm, this time by executing randomly generated circuits. There's no hard proof that something based on these principles could not work, but the computer hardware experts that we reached out to in 2014 tended to be fairly pessimistic on it. Matthew Wampler-Doty himself suggested a proof of work based on SAT solving, but this too was ultimately rejected.

Finally, we came full circle with an algorithm called "Dagger Hashimoto". "Dashimoto", as it was sometimes called in short, borrowed many ideas from Hashimoto, a proof of work algorithm by Thaddeus Dryja that pioneered the notion of "I/O bound proof of work", where the dominant limiting factor in mining speed was not hashes per second, but rather megabytes per second of RAM access. However, it combined this with Dagger's notion of light-client-friendly DAG-generated datasets. After many rounds of tweaking by myself, Matthew, Tim and others, the ideas finally converged into the algorithm we now call Ethash.

```python
def hashimoto(header, nonce, full_size, dataset_lookup):
    n = full_size / HASH_BYTES
    w = MIX_BYTES // WORD_BYTES
    mixhashes = MIX_BYTES / HASH_BYTES
    # combine header+nonce into a 64 byte seed
    s = sha3_512(header + nonce[::-1])
    # start the mix with replicated s
    mix = []
    for _ in range(MIX_BYTES / HASH_BYTES):
        mix.extend(s)
    # mix in random dataset nodes
    for i in range(ACCESSES):
        p = fnv(i ^ s[0], mix[i % w]) % (n // mixhashes) * mixhashes
        newdata = []
        for j in range(MIX_BYTES / HASH_BYTES):
            newdata.extend(dataset_lookup(p + j))
        mix = map(fnv, mix, newdata)
    # compress mix
    cmix = []
    for i in range(0, len(mix), 4):
        cmix.append(fnv(fnv(fnv(mix[i], mix[i+1]), mix[i+2]), mix[i+3]))
    return {
        "mix digest": serialize_hash(cmix),
        "result": serialize_hash(sha3_256(s+cmix))
    }

def hashimoto_light(full_size, cache, header, nonce):
    return hashimoto(header, nonce, full_size, lambda x: calc_dataset_item(cache, x))

def hashimoto_full(full_size, dataset, header, nonce):
    return hashimoto(header, nonce, full_size, lambda x: dataset[x])
```

By the summer of 2014, the protocol had considerably stabilized, with the major exception of the proof of work algorithm which would not reach the Ethash phase until around the beginning of 2015, and a semi-formal specification existed in the form of Gavin's yellow paper.

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated, through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make

information is often lacking, and plain old prejudices are difficult to shake.

Overall, I wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

In August 2014, I developed and introduced the uncle mechanism, which allows Ethereum's blockchain to have a shorter block time and higher capacity while mitigating centralization risks. This was introduced as part of PoC6.

Discussions with the Bitshares team led us to consider adding heaps as a first-class data structure, though we ended up not doing this due to lack of time, and later security audits and DoS attacks will show that it is actually much harder than we had thought at the time to do this safely.

In September, Gavin and I planned out the next two major changes to the protocol design. First, alongside the state tree and transaction tree, every block would also contain a "receipt tree". The receipt tree would include hashes of the logs created by a transaction, along with intermediate state roots. Logs would allow transactions to create "outputs" that are saved in the blockchain, and are accessible to light clients, but that are not accessible to future state calculations. This could be used to allow decentralized applications to easily query for events, such as token transfers, purchases, exchange orders being created and filled, auctions being started, and so forth.

There were other ideas that were considered, like making a Merkle tree out of the entire execution trace of a transaction to allow anything to be proven; logs were chosen because they were a compromise between simplicity and completeness.

The second was the idea of "precompiles", solving the problem of allowing complex cryptographic computations to be usable in the EVM without having to deal with EVM overhead. We had also gone through many more ambitious ideas about "native contracts", where if miners have an optimized implementation of some contracts they could "vote" the gasprice of those contracts down, so contracts that most miners could execute much more quickly would naturally have a lower gas price; however, all of these ideas were rejected because we could not come up with a cryptoeconomically safe way to implement such a thing. An attacker could always create a contract which executes some trapdoored cryptographic operation, distribute the trapdoor to themselves and their friends to allow them to execute this contract much faster, then vote the gasprice down and use this to DoS the network. Instead we opted for the much less ambitious approach of having a smaller number of precompiles that are simply specified in the protocol, for common operations such as hashes and signature schemes.

Gavin was also a key initial voice in developing the idea of "protocol abstraction" - moving as many parts of the protocol such as ether balances, transaction signing algorithms, nonces, etc into the protocol itself as contracts, with a theoretical final goal of reaching a situation where the entire ethereum protocol could be described as making a function call into a virtual machine that has some pre-initialized state. There was not enough time for these ideas to get into the initial Frontier release, but the principles are expected to start slowly getting integrated through some of the Constantinople changes, the Casper contract and the sharding specification.

This was all implemented in PoC7; after PoC7, the protocol did not really change much, with the exception of minor, though in some cases important, details that would come out through security audits...

In early 2015, came the pre-launch security audits organized by Jutta Steiner and others, which included both software code audits and academic audits. The software audits were primarily on the C++ and Go implementations, which were led by Gavin Wood and Jeffrey Wilcke, respectively, though there was also a smaller audit on my pyethereum implementation. Of the two academic audits, one was performed by Ittay Eyal (of "selfish mining" fame), and the other by Andrew Miller and others from Least Authority. The Eyal audit led to a minor protocol change: the total difficulty of a chain would not include uncles. The Least Authority audit was more focused on smart contract and gas economics, as well as the Patricia tree. This audit led to several protocol changes. One small one is the use of sha3(addr) and sha3(key) as trie keys instead of the address and key directly; this would make it harder to perform a worst-case attack on the trie.

There are useful parallels between this refund loop and the publish-subscribe function illustrated in Miller's thesis. He demonstrates several hazards that are present when the `publish` callbacks are run synchronously:

- exceptions raised during the callback would prevent execution of later callbacks
- reentrancy hazards if the callback itself executes publish(), subscribe(), or unsubscribe(): repeated actions, missing actions, and inconsistent delivery of messages

Some analogous issues in the crowdfund example are:

- delivering a contribution, after the funding deadline, with just enough gas to allow some refunds to go through, but not all: the contract could be left in a state where it was unable to refund the remaining contributions
- if the refund was triggered by a contract at the end of a long call stack, the `send` instructions will fail. However the example appears to ignore the return value of the `send`, so execution will continue. All records will be cleared, and the funds can never be recovered.
- the refund callback could make a new donation, triggering another refund cycle, potentially double-refunding the earlier contributions, or failing to refund later ones

And a warning that was perhaps a bit too far ahead of its time...

Another significant thing that we discussed was the gas limit voting mechanism. At the time, we were already concerned by perceived lack of progress in the bitcoin block size debate, and wanted to have a more flexible design in Ethereum that could adjust over time as needed. But the challenge is: what is the optimal limit? My initial thought had been to make a dynamic limit, targeting $1.5 \cdot$ the long-term exponential moving average of the actual gas usage, so that in the long run on average blocks would be $\frac{2}{3}$ full. However, Andrew showed that this was exploitable in some ways - specifically, miners who wanted to raise the limit would simply include transactions in their own blocks that consume a very large amount of gas, but take very little time to process, and thereby always create full blocks at no cost to themselves. The security model was thus, at least in the upward direction, equivalent to simply having miners vote on the gas limit.

We did not manage to come up with a gas limit strategy that was less likely to break, and so Andrew's recommended solution was to simply have miners vote on the gas limit explicitly, and have the default strategy for voting be the $1.5 \cdot$ EMA rule. The reasoning was that we were still very far from knowing the right approach for setting maximum gas limits, and the risk of any specific approach failing seemed greater than the risk of miners abusing their voting power. Hence, we might as well simply let miners vote on the gas limit, and accept the risk that the limit will go too high or too low, in exchange for the benefit of flexibility, and the ability for miners to work together to very quickly adjust the limit upwards or downwards as needed.
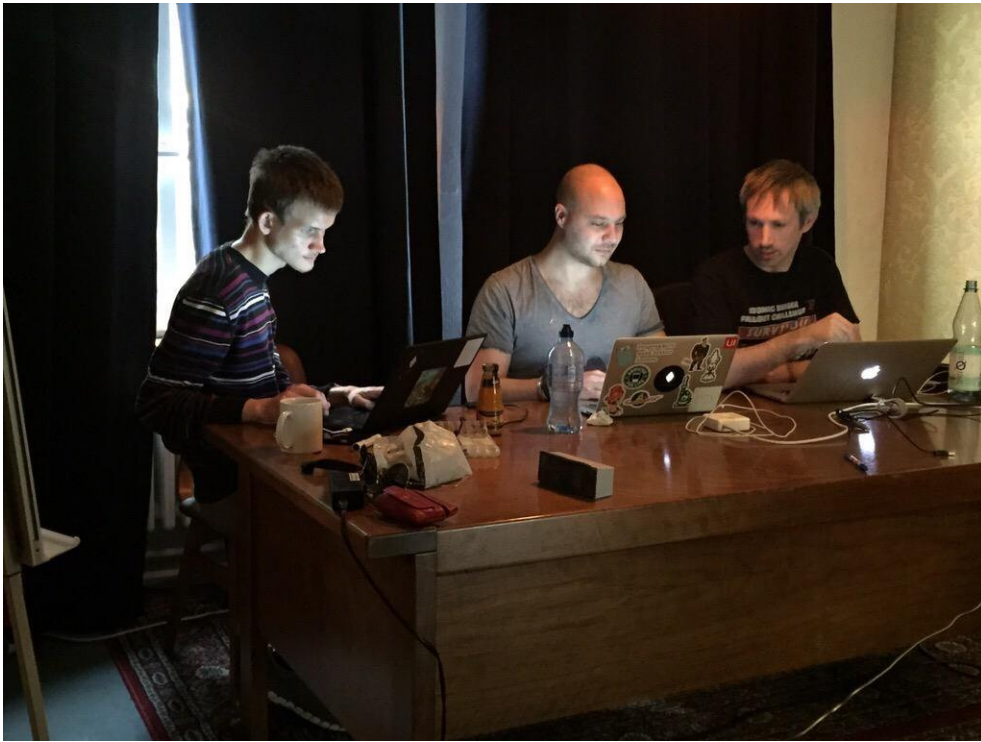
After a mini-hackathon between Gavin, Jeff and myself, PoC9 was launched in March, and was intended to be the final proof of concept release. A testnet, Olympic, ran for four months, using the protocol that was intended to be used in the livenet, and Ethereum's long-term plan was established. Vinay Gupta wrote a blog post, "The Ethereum Launch Process", that described the four expected stages of Ethereum livenet development, and gave them their current names: Frontier, Homestead, Metropolis and Serenity.

Olympic ran for four months. In the first two months, many bugs were found in the various implementations, consensus failures happened, among other issues, but around June the network noticeably stabilized. In July a decision was made to make a code-freeze, followed by a release, and on July 30 the release took place.

A Note on Metcalfe's Law, Externalities and Ecosystem Splits

2017 Jul 27

Looks like it's blockchain split season [again](). For background of various people discussing the topic, and whether such splits are good or bad:

- Power laws and network effects (arguing the BTC/BCC split may destroy value due to network effect loss): https://medium.com/crypto-fundamental/power-laws-and-network-effects-why-bitcoincash-is-not-a-free-lunch-5adb579972aa
- Brian Armstrong on the Ethereum Hard Fork (last year): https://blog.coinbase.com/on-the-ethereum-hard-fork-780f1577e986
- Phil Daian on the ETH/ETC split: http://pdaian.com/blog/stop-worrying-love-etc/

Given that ecosystem splits are not going away, and we may well see more of them in the crypto industry over the next decade, it seems useful to inform the discussion with some simple economic modeling. With that in mind, let's get right to it.

Suppose that there exist two projects A and B, and a set of users of total size N, where A has N_a users and B has N_b users. Both projects benefit from network effects, so they have a utility that increases with the number of users. However, users also have their own differing taste preferences, and this may lead them to choose the smaller platform over the bigger platform if it suits them better.

We can model each individual's private utility in one of four ways:

1. $U(A) = p + N_a$ $U(B) = q + N_b$
2. $U(A) = p \cdot N_a$ $U(B) = q \cdot N_b$
3. $U(A) = p + \ln{N_a}$ $U(B) = q + \ln{N_b}$
4. $U(A) = p \cdot \ln{N_a}$ $U(B) = q \cdot \ln{N_b}$

p and q are private per-user parameters that you can think of as corresponding to users' distinct preferences. The difference between the first two approaches and the last two reflects differences between interpretations of Metcalfe's law, or more broadly the idea that the per-user value of a system grows with the number of users. The [original formulation]() suggested a per-user value of N (that is, a total network value of N^{2}), but other analysis (see [here]()) suggests that above very small scales $N\log{N}$ usually dominates; there is a controversy over which model is correct. The difference between the first and second (and between the third and fourth) is the extent to which utility from a system's intrinsic quality and utility from network effects are complementary - that is, are the two things good in completely separate ways that do not interact with each other, like social media and coconuts, or are network effects an important part of letting the intrinsic quality of a system shine?

We can now analyze each case in turn by looking at a situation where N_a users choose A and N_b users choose B, and analyze each individual's decision to choose one or the other from the perspective of economic externalities - that is, does a user's choice to switch from A to B have a positive net effect on others' utility or a negative one? If switching has a positive externality, then it is virtuous and should be socially nudged or encouraged, and if it has a negative externality then it should be discouraged. We model an "ecosystem split" as a game where to start off $N_a = N$ and $N_b = 0$ and users are deciding for themselves whether or not to join the split, that is, to move from A to B, possibly causing N_a to fall and N_b to rise.

Switching (or not switching) from A to B has externalties because A and B both have network effects; switching from A to B has the negative externality of reducing A's network effect, and so hurting all remaining A users, but it also has the positive externality of increasing B's network effect, and so benefiting all B users.

Case 1

Switching from A to B gives N_a users a negative externality of one, so a total loss of N_a, and it gives N_b users a positive externality of one, so a total gain of N_b. Hence, the total externality is of size $N_b - N_a$; that is, switching from the smaller to the larger platform has positive externalities, and switching from the larger platform to the smaller platform has negative externalities.

Case 2

Suppose P_a is the sum of p values of N_a users, and Q_b is the sum of q values of N_b users. The total negative externality is P_a and the total positive externality is Q_b. Hence, switching from the smaller platform to the larger has positive social externalities if the two platforms have equal intrinsic quality to their users (ie. users of A intrinsically enjoy A as much as users of B intrinsically enjoy B, so p and q values are evenly distributed), but if it is the case that A is bigger but B is better, then there are positive externalities in switching to B.

Furthermore, notice that if a user is making a switch from a larger A to a smaller B, then this itself is revealed-preference evidence that, for that user, and for all existing users of B, $\frac{q}{p} > \frac{N_a}{N_b}$. However, if the split stays as a split, and does not proceed to become a full-scale migration, then that means that users of A hold different views, though this could be for two reasons: (i) they intrinsically dislike A but not by enough to justify the switch, (ii) they intrinsically like A more than B. This could arise because (a) A users have a higher opinion of A than B users, or (b) A users have a lower opinion of B than B users. In general, we see that moving from a system that makes its average user less happy to a system that makes its average user more happy has positive externalities, and in other situations it's difficult to say.

Case 3

The derivative of $\ln{x}$ is $\frac{1}{x}$. Hence, switching from A to B gives N_a users a negative externality of $\frac{1}{N_a}$, and it gives N_b users a positive externality of $\frac{1}{N_b}$. Hence, the negative and positive externalities are both of total size one, and thus cancel out. Hence, switching from one platform to the other imposes no social externalities, and it is socially optimal if all users switch from A to B if and only if they think that it is a good idea for them personally to do so.

Case 4

Let P_a and Q_b are before. The negative externality is of total size $\frac{P_a}{N_a}$ and the positive externality is of total size $\frac{Q_b}{N_b}$. Hence, if the two systems have equal intrinsic quality, the externality is of size zero, but if one system has higher intrinsic quality, then it is virtuous to switch to it. Note that as in case 2, if users *are* switching from a larger system to a smaller system, then that means that they find the smaller system to have higher intrinsic quality, although, also as in case 2, if the split remains a split and does not become a full-scale migration, then that means other users see the intrinsic quality of the larger system as higher, or at least not lower by enough to be worth the network effects.

The existence of users switching to B suggests that for them, $\frac{q}{p} \geq \frac{\log{N_a}}{\log{N_b}}$, so for the $\frac{Q_B}{N_b} > \frac{P_a}{N_a}$ condition to not hold (ie. for a move from a larger system to a smaller system not to have positive externalities) it would need to be the case that users of A have similarly high values of p - an approximate heuristic is, the users of A would need to love A so much that if *they* were the ones in the minority that would be willing to split off and move to (or stay with) the smaller system. In general, it thus seems that moves from larger systems to smaller systems that actually do happen will have positive externalities, but it is far from ironclad that this is the case.

Hence, if the first model is true, then to maximize social welfare we should be trying to nudge people to switch to (or stay with) larger systems over smaller systems, and splits should be discouraged. If the fourth model is true, then we should be at least slightly trying to nudge people to switch to smaller systems over larger systems, and splits should be slightly encouraged. If the third model is true, then people will choose the socially optimal thing all by themselves, and if the second model is true, it's a toss-up.

It is my personal view that the truth lies somewhere between the third and fourth models, and the first and second greatly overstate network effects above small scales. The first and second model (the N^{2} form of Metcalfe's law) essentially state that a system growing from 990 million to 1 billion users gives the same increase in per-user utility as growing from 100,000 to 10.1 million users, which seems very unrealistic, whereas the $N\log{N}$ model (growing from 100 million to 1 billion users gives the same increase in per-user utility as growing from 100,000 to 10 million users) intuitively seems much more correct.

And the third model says: if you see people splitting off from a larger system to create a smaller system because they want something that more closely matches their personal values, then the fact that these people have already shown that they value this switch enough to give up the comforts of the original system's network effects is by itself enough evidence to show that the split is socially beneficial. Hence, unless I can be convinced that the first model is true, or that the second model is true and the specific distributions of p and q values make splits make negative negative externalities, I maintain my existing view that those splits that actually do happen (though likely not *hypothetical* splits that end up not happening due to lack of interest) are in the long term socially beneficial, value-generating events.

The Triangle of Harm

2017 Jul 16

The following is a diagram from a slide that I made in one of my presentations at Cornell this week:
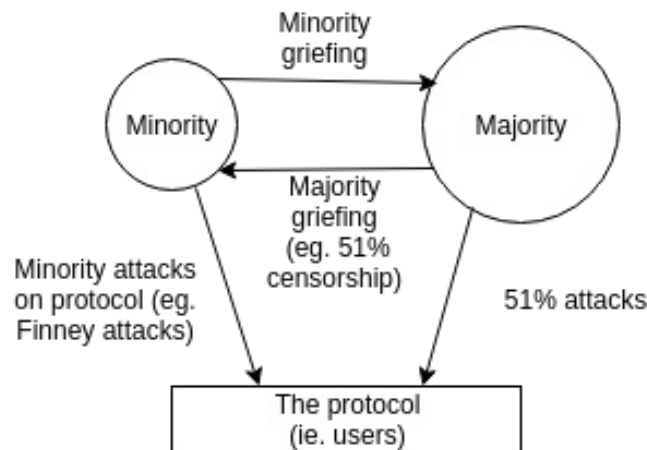
If there was one diagram that could capture the core principle of Casper's incentivization philosophy, this might be it. Hence, it warrants some further explanation.

The diagram shows three constituencies - the minority, the majority and the protocol (ie. users), and four arrows representing possible adversarial actions: the minority attacking the protocol, the minority attacking the majority, the majority attacking the protocol, and the majority attacking the minority. Examples of each include:

- **Minority attacking the protocol** - [Finney attacks](#) (an attack done by a miner on a proof of work blockchain where the miner double-spends unconfirmed, or possibly single-confirmed, transactions)
- **Minority attacking the majority** - [feather forking](#) (a minority in a proof of work chain attempting to revert any block that contains some undesired transactions, though giving up if the block gets two confirmations)
- **Majority attacking the protocol** - traditional 51% attacks
- **Majority attacking the minority** - a 51% censorship attack, where a cartel refuses to accept any blocks from miners (or validators) outside the cartel

The essence of Casper's philosophy is this: **for all four categories of attack**, we want to put an upper bound on the ratio between the amount of harm suffered by the victims of the attack and the cost to the attacker. In some ways, every design decision in Casper flows out of this principle.

This differs greatly from the usual proof of work incentivization school of thought in that in the proof of work view, the last two attacks are left undefended against. The first two attacks, Finney attacks and feather forking, are costly because the attacker risks their blocks not getting included into the chain and so loses revenue. If the attacker is a majority, however, the attack is costless, because the attacker can always guarantee that their chain will be the main chain. In the long run, difficulty adjustment ensures that the total revenue of all miners is exactly the same no matter what, and this further means that if an attack causes some victims to lose money, then the attacker *gains* money.

This property of proof of work arises because traditional Nakamoto proof of work fundamentally *punishes dissent* - if you as a miner make a block that aligns with the consensus, you get rewarded, and if you make a block that does not align with the consensus you get penalized (the penalty is not in the protocol; rather, it comes from the fact that such a miner expends electricity and capital to produce the block and gets no reward).

Casper, on the other hand, works primarily by *punishing equivocation* - if you send two messages that conflict with each other, then you get very heavily penalized, even if one of those messages aligns with the consensus (read more on this in the [blog post on "minimal slashing conditions"](#)).

Hence, in the event of a finality reversion attack, those who caused the reversion event are penalized, and everyone else is left untouched; the majority can attack the protocol only at heavy cost, and the majority cannot cause the minority to lose money.

It gets more challenging when we move to talking about two other kinds of attacks - liveness faults, and censorship. A liveness fault is one where a large portion of Casper validators go offline, preventing the consensus from reaching finality, and a censorship fault is one where a majority of Casper validators refuse to accept some transactions, or refuse to accept consensus messages from other Casper validators (the victims) in order to deprive them of rewards.

This touches on a fundamental dichotomy: **speaker/listener fault equivalence**.

Suppose that person B says that they did not receive a message from person A. There are two possible explanations: (i) person A did not send the message, (ii) person B pretended not to hear the message. Given just the evidence of B's claim, *there is no way to tell which of the two explanations is correct*. The relation to blockchain protocol incentivization is this: if you see a protocol execution where 70% of validators' messages are included in the chain and 30% are not, and see nothing else (and this is what the blockchain sees), then there is no way to tell whether the problem is that 30% are offline or 70% are censoring. If we want to make both kinds of attacks expensive, there is only one thing that we can do: **penalize both sides**.

Penalizing both sides allows either side to "grief" the other, by going offline if they are a minority and censoring if they are a majority. However, we can establish bounds on how easy this griefing is, through the technique of **griefing factor analysis**. The griefing factor of a strategy is essentially the amount of money lost by the victims divided by the amount of money lost by the attackers, and the griefing factor of a protocol is the highest griefing factor that it allows. For example, if a protocol allows me to cause you to lose $3 at a cost of $1 to myself, then the griefing factor is 3. If there are

no ways to cause others to lose money, the griefing factor is zero, and if you can cause others to lose money at no cost to yourself (or at a benefit to yourself), the griefing factor is infinity.

In general, **wherever a speaker/listener dichotomy exists, the griefing factor cannot be globally bounded above by any value below 1**. The reason is simple: either side can grief the other, so if side A can grief side B with a factor of x then side B can grief side A with a factor of $\frac{1}{x}$; x and $\frac{1}{x}$ cannot both be below 1 simultaneously. We can play around with the factors; for example, it may be considered okay to allow griefing factors of 2 for majority attackers in exchange for keeping the griefing factor at 0.5 for minorities, with the reasoning that minority attackers are more likely. We can also allow griefing factors of 1 for small-scale attacks, but specifically for large-scale attacks force a chain split where on one chain one side is penalized and on another chain another side is penalized, trusting the market to pick the chain where attackers are not favored. Hence there is a lot of room for compromise and making tradeoffs between different concerns within this framework.

Penalizing both sides has another benefit: it ensures that if the protocol is harmed, the attacker is penalized. This ensures that whoever the attacker is, they have an incentive to avoid attacking that is commensurate with the amount of harm done to the protocol. However, if we want to bound the ratio of harm to the protocol over cost to attackers, we need a formalized way of measuring how much harm to the protocol was done.

This introduces the concept of the **protocol utility function** - a formula that tells us how well the protocol is doing, that should ideally be calculable from inside the blockchain. In the case of a proof of work chain, this could be the percentage of all blocks produced that are in the main chain. In Casper, protocol utility is zero for a perfect execution where every epoch is finalized and no safety failures ever take place, with some penalty for every epoch that is not finalized, and a very large penalty for every safety failure. If a protocol utility function can be formalized, then penalties for faults can be set as close to the loss of protocol utility resulting from those faults as possible.

On Path Independence

2017 Jun 22

Suppose that someone walks up to you and starts exclaiming to you that he thinks he has figured out how to create a source of unlimited free energy. His scheme looks as follows. First, you get a spaceship up to low Earth orbit. There, Earth's gravity is fairly high, and so the spaceship will start to accelerate heavily toward the earth. The spaceship puts itself into a trajectory so that it barely brushes past the Earth's atmosphere, and then keeps hurtling far into space. Further in space, the gravity is lower, and so the spaceship can go higher before it starts once again coming down. When it comes down, it takes a curved path toward the Earth, so as to maximize its time in low orbit, maximizing the acceleration it gets from the high gravity there, so that after it passes by the Earth it goes even higher. After it goes high enough, it flies through the Earth's atmosphere, slowing itself down but using the waste heat to power a thermal reactor. Then, it would go back to step one and keep going.

Something like this:
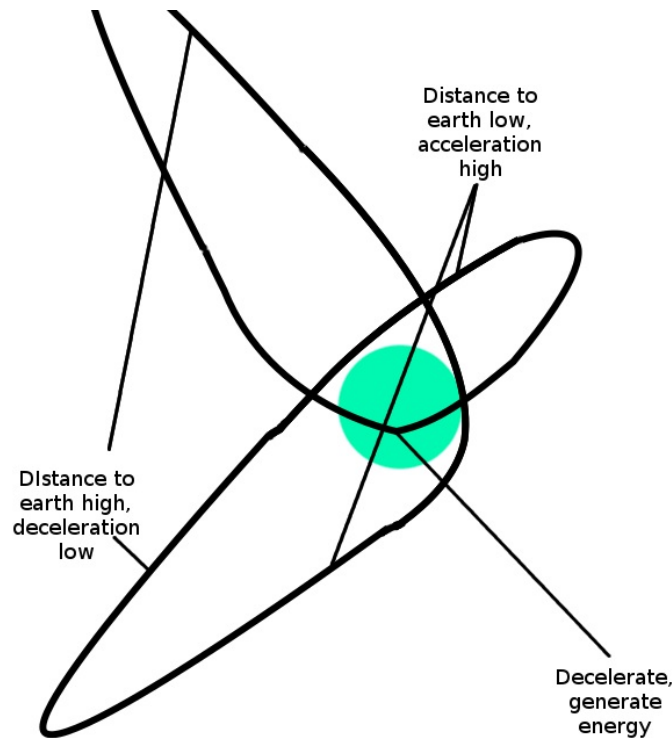
Now, if you know anything about Newtonian dynamics, chances are you'll immediately recognize that this scheme is total bollocks. But how do you know? You could make an appeal to symmetry, saying "look, for every slice of the orbital path where you say gravity gives you high acceleration, there's a corresponding slice of the orbital path where gravity gives you just as high deceleration, so I don't see where the net gains are coming from". But then, suppose the man presses you. "Ah," he says, "but in that slice where there is high acceleration your initial velocity is low, and so you spend a lot of time inside of it, whereas in the corresponding slice, your incoming velocity is high, and so you have less time to decelerate". How do you really, conclusively, prove him wrong?

One approach is to dig deeply into the math, calculate the integrals, and show that the supposed net gains are in fact exactly equal to zero. But there is also a simple approach: recognize that **energy is path-independent**. That is, when the spaceship moves from point A to point B, where point B is closer to the earth, its kinetic energy certainly goes up because its speed increases. But because total energy (kinetic plus potential) is conserved, and potential energy is only dependent on the spaceship's *position*, and not how it got there, we know that regardless of what path from point

$(A\)$ to point $\(B\)$ the spaceship takes, once it gets to point $\(B\)$ *the total change in kinetic energy will be exactly the same*.
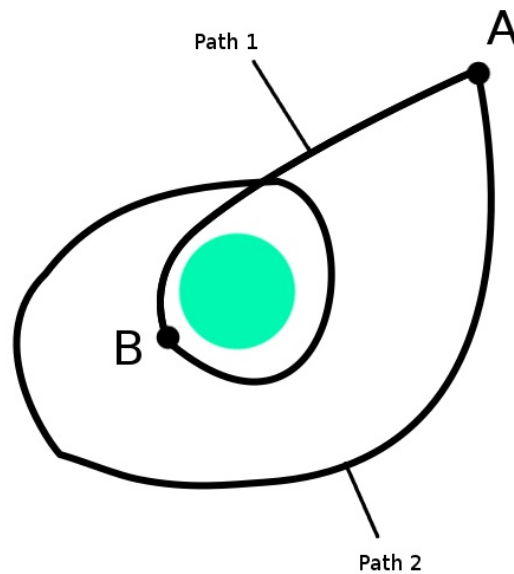
Different paths, same change in energy

Furthermore, we know that the kinetic energy gain from going *from point $\(A\)$ to point $\(A\)$* is also independent of the path you take along the way: in all cases it's exactly zero.
One [concern](#) sometimes [cited](#) against on-chain market makers (that is, fully automated on-chain mechanisms that act as always-available counterparties for people who wish to trade one type of token for another) is that they are invariably easy to exploit.

As an example, let me quote a [recent post](#) discussing this issue in the context of Bancor:

> The prices that Bancor offers for tokens have nothing to do with the actual market equilibrium. Bancor will always trail the market, and in doing so, will bleed its reserves. A simple thought experiment suffices to illustrate the problem. Suppose that market panic sets around X. Unfounded news about your system overtake social media. Let's suppose that people got convinced that your CEO has absconded to a remote island with no extradition treaty, that your CFO has been embezzling money, and your CTO was buying drugs from the darknet markets and shipping them to his work address to make a Scarface-like mound of white powder on his desk. Worse, let's suppose that you know these allegations to be false. They were spread by a troll army wielded by a company with no products, whose business plan is to block everyone's coin stream. Bancor would offer ever decreasing prices for X coins during a bank run, until it has no reserves left. You'd watch the market panic take hold and eat away your reserves. Recall that people are convinced that the true value of X is 0 in this scenario, and the Bancor formula is guaranteed to offer a price above that. So your entire reserve would be gone.

The post discusses many issues around the Bancor protocol, including details such as code quality, and I will not touch on any of those; instead, I will focus purely on the topic of on-chain market maker efficiency and exploitability, using Bancor (along with MKR) purely as examples and not seeing to make any judgements on the quality of either project as a whole.

For many classes of naively designed on-chain market makers, the comment above about exploitability and trailing markets applies verbatim, and quite seriously so. However, there are also classes of on-chain market makers that are definitely not suspect to their entire reserve being drained due to some kind of money-pumping attack. To take a simple example, consider the market maker selling MKR for ETH whose internal state consists of a current price, $\(p\)$, and which is willing to buy or sell an infinitesimal amount of MKR at each price level. For example, suppose that $\(p = 5\)$, and you wanted to buy 2 MKR. The market would sell you:

- 0.00...01 MKR at a price of 5 ETH/MKR
- 0.00...01 MKR at a price of 5.00...01 ETH/MKR
- 0.00...01 MKR at a price of 5.00...02 ETH/MKR

- ....
- 0.00...01 MKR at a price of 6.99...98 ETH/MKR
- 0.00...01 MKR at a price of 6.99...99 ETH/MKR

Altogether, it's selling you 2 MKR at an average price of 6 ETH/MKR (ie. total cost 12 ETH), and at the end of the operation p has increased to 7. If someone *then* wanted to sell 1 MKR, they would be spending 6.5 ETH, and at the end of *that* operation p would drop back down to 6.

Now, suppose that I told you that such a market maker started off at a price of $p = 5$, and after an unspecified series of events p is now 4. Two questions:

1. How much MKR did the market maker gain or lose?
2. How much ETH did the market maker gain or lose?

The answers are: it gained 1 MKR, and lost 4.5 ETH. Notice that this result is totally independent of the path that p took. Those answers are correct if p went from 5 to 4 directly with one buyer, they're correct if there was first one buyer that took p from 5 to 4.7 and a second buyer that took p the rest of the way to 4, and they're even correct if p first dropped to 2, then increased to 9.818, then dropped again to 0.53, then finally rose again to 4.

Why is this the case? The simplest way to see this is to see that if p drops below 4 and then comes back up to 4, the sells on the way down are exactly counterbalanced by buys on the way up; each sell has a corresponding buy of the same magnitude at the exact same price. But we can also see this by viewing the market maker's core mechanism differently. Define the market maker as having a *single-dimensional* internal state p, and having MKR and ETH balances defined by the following formulas:

$mkr\_balance(p) = 10 - p$

$eth\_balance(p) = p^2/2$

Anyone has the power to "edit" p (though only to values between 0 and 10), but they can only do so by supplying the right amount of MKR or ETH, and getting the right amount of MKR and ETH back, so that the balances still match up; that is, so that the amount of MKR and ETH held by the market maker after the operation is the amount that they are supposed to hold according to the above formulas, with the new value for p that was set. Any edit to p that does not come with MKR and ETH transactions that make the balances match up automatically fails.

Now, the fact that any series of events that drops p from 5 to 4 also raises the market maker's MKR balance by 1 and drops its ETH balance by 4.5, regardless of what series of events it was, should look elementary: $mkr\_balance(4) - mkr\_balance(5) = 1$ and $eth\_balance(4) - eth\_balance(5) = -4.5$.

What this means is that a "reserve bleeding" attack on a market maker that preserves this kind of path independence property is impossible. Even if some trolls successfully create a market panic that drops prices to near-zero, when the panic subsides, and prices return to their original levels, the market maker's position will be unchanged - even if both the price, and the market maker's balances, made a bunch of crazy moves in the meantime.

Now, this does not mean that market makers cannot lose money, compared to other holding strategies. If, when you start off, 1 MKR = 5 ETH, and then the MKR price moves, and we compare the performance of holding 5 MKR and 12.5 ETH in the market maker versus the performance of just holding the assets, the result looks like this:

Return of market maker versus balanced portfolio
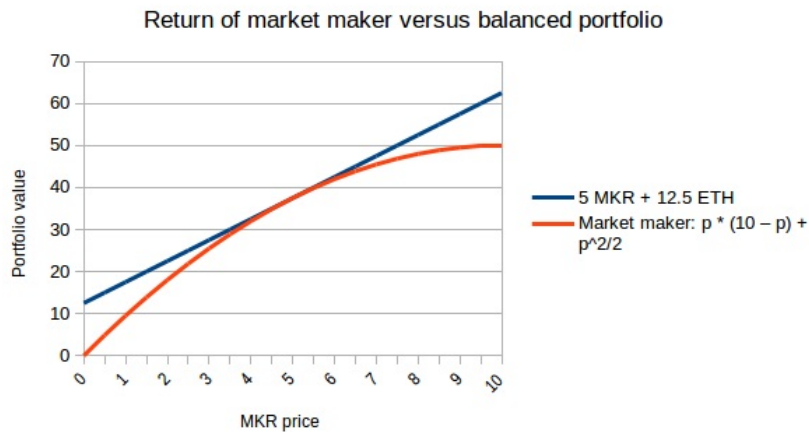
Holding a balanced portfolio always wins, except in the case where prices stay exactly the same, in which case the returns of the market maker and the balanced portfolio are equal. Hence, the purpose of a market maker of this type is to subsidize guaranteed liquidity as a public good for users, serving as trader of last resort, and not to earn revenue. However, we certainly can modify the market maker to earn revenue, and quite simply: we have it charge a spread. That is, the market maker might charge $1.005\cdot p$ for buys and offer only $0.995\cdot p$ for sells. Now, being the beneficiary of a market maker becomes a bet: if, in the long run, prices tend to move in one direction, then the market maker loses, at least relative to what they could have gained if they had a balanced portfolio. If, on the other hand, prices tend to bounce around wildly but ultimately come back to the same point, then the market maker can earn a nice profit. This sacrifices the "path independence" property, but in such a way that any deviations from path independence are always in the market maker's favor.

There are many designs that path-independent market makers could take; if you are willing to create a token that can issue an unlimited quantity of units, then the "constant reserve ratio" mechanism (where for some constant ratio $0 \leq r \leq 1$, the token supply is $p^{1/r - 1}$ and the reserve size is $r \cdot p^{1/r}$) also counts as one, provided that it is implemented correctly and path independence is not compromised by bounds and rounding errors.

If you want to make a market maker for existing tokens without a price cap, my favorite (credit to Martin Koppelmann) mechanism is that which maintains the invariant $tokenA\_balance(p) \cdot tokenB\_balance(p) = k$ for some constant k. So the formulas would be:

$tokenA\_balance(p) = \sqrt{k\cdot p}$

$tokenB\_balance(p) = \sqrt{k/p}$
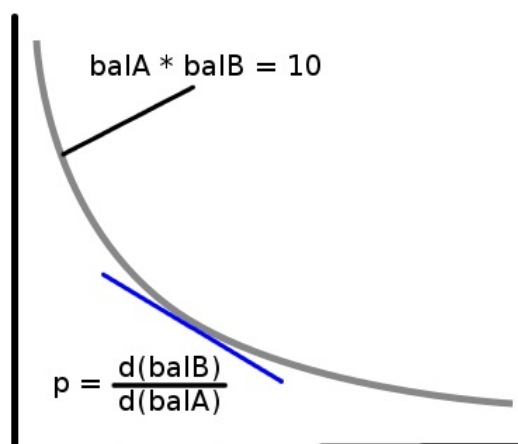
Where p is the price of $tokenB$ denominated in $tokenA$. In general, you can make a path-independent market maker by defining any (monotonic) relation between $tokenA\_balance$ and $tokenB\_balance$ and calculating its derivative at any point to give the price.

The above only discusses the role of path independence in preventing one particular type of issue: that where an attacker somehow makes a series of transactions in the context of a series of price movements in order to repeatedly drain the market maker of money. With a path independent market maker, such "money pump" vulnerabilities are impossible. However, there certainly are *other* kinds of inefficiencies that may exist. If the price of MKR drops from 5 ETH to 1 ETH, then the market maker used in the example above will have lost 28 ETH worth of value, whereas a balanced portfolio would only have lost 20 ETH. Where did that 8 ETH go?

In the best case, the price (that is to say, the "real" price, the price level where supply and demand among all users and traders matches up) drops quickly, and some lucky trader snaps up the deal, claiming an 8 ETH profit minus negligible transaction fees. But what if there are multiple traders? Then, if the price between block n and block $n+1$ differs, the fact that traders can bid against each other by setting transaction fees creates an all-pay auction, with revenues going to the miner. As a consequence of the [revenue equivalence theorem](#), we can deduce that we can expect that the transaction fees that traders send into this mechanism will keep going up until they are roughly equal to the size of the profit earned (at least initially; the *real* equilibrium is for miners to just snap up the money themselves). Hence, either way schemes like this are ultimately a gift to the miners.

One way to increase social welfare in such a design is to make it possible to create purchase transactions that are only worthwhile for miners to include if they actually make the purchase. That is, if the "real" price of MKR falls from 5 to 4.9, and there are 50 traders racing to arbitrage the market maker, and only the first one of those 50 will make the trade, then only that one should pay the miner a transaction fee. This way, the other 49 failed trades will not clog up the blockchain. [EIP 86](#), slated for Metropolis, opens up a path toward standardizing such a conditional transaction fee mechanism (another good side effect is that this can also make token sales more unobtrusive, as similar all-pay-auction mechanics apply in many token sales).

Additionally, there are other inefficiencies if the market maker is the *only* available trading venue for tokens. For example, if two traders want to exchange a large amount, then they would need to do so via a long series of small buy and sell transactions, needlessly clogging up the blockchain. To mitigate such efficiencies, an on-chain market maker should only be *one* of the trading venues available, and not the only one. However, this is arguably not a large concern for protocol developers; if there ends up being a demand for a venue for facilitating large-scale trades, then someone else will likely provide it.

Furthermore, the arguments here only talk about path independence of the market maker *assuming* a given starting price and ending price. However, because of various psychological effects, as well as multi-equilibrium effects, the ending price is plausibly a function not just of the starting price and recent events that affect the "fundamental" value of the asset, but also of the pattern of trades that happens in response to those events. If a price-dropping event takes place, and because of poor liquidity the price of the asset drops quickly, it may end up recovering to a lower point than if more liquidity had been present in the first place. That said, this may actually be an argument in favor of subsidied market makers: if such multiplier effects exist, then they will have a positive impact on price stability that goes beyond the first-order effect of the liquidity that the market maker itself provides.

There is likely a lot of research to be done in determining exactly which path-independent market maker is optimal. There is also the possibility of hybrid semi-automated market makers that have the same guaranteed-liquidity properties, but which include some element of asynchrony, as well as the ability for the operator to "cut in line" and collect the profits in cases where large amounts of capital would otherwise be lost to miners. There is also not yet a coherent theory of just how much (if any) on-chain automated guaranteed liquidity is optimal for various objectives, and to what extent, and by whom, these market makers should be subsidized. All in all, the on-chain mechanism design space is still in its early days, and it's certainly worth much more broadly researching and exploring various options.

Analyzing Token Sale Models

2017 Jun 09

Note: I mention the names of various projects below only to compare and contrast their token sale mechanisms; this should NOT be taken as an endorsement or criticism of any specific project as a whole. It's entirely possible for any given project to be total trash as a whole and yet still have an awesome token sale model.

The last few months have seen an increasing amount of innovation in token sale models. Two years ago, the space was simple: there were capped sales, which sold a fixed number of tokens at a fixed price and hence fixed valuation and would often quickly sell out, and there were uncapped sales, which sold as many tokens as people were willing to buy. Now, we have been seeing a surge of interest, both in terms of theoretical investigation and in many cases real-world implementation, of hybrid capped sales, reverse dutch auctions, Vickrey auctions, proportional refunds, and many other mechanisms.

Many of these mechanisms have arisen as responses to perceived failures in previous designs. Nearly every significant sale, including Brave's Basic Attention Tokens, Gnosis, upcoming sales such as Bancor, and older ones such as Maidsafe and even the Ethereum sale itself, has been met with a substantial amount of criticism, all of which points to a simple fact: so far, we have still not yet discovered a mechanism that has all, or even most, of the properties that we would like.

Let us review a few examples.

Maidsafe

The decentralized internet platform raised $7m in five hours. However, they made the mistake of accepting payment in two currencies (BTC and MSC), and giving a favorable rate to MSC buyers. This [led to](led to) a temporary ~2x appreciation in the MSC price, as users rushed in to buy MSC to participate in the sale at the more favorable rate, but then the price saw a similarly steep drop after the sale ended. Many users converted their BTC to MSC to participate in the sale, but then the sale closed too quickly for them, leading to them being stuck with a ~30% loss.

This sale, and several others after it (cough cough WeTrust, TokenCard), showed a lesson that should hopefully by now be uncontroversial: running a sale that accepts multiple currencies at a fixed exchange rate is dangerous and bad. Don't do it.

Ethereum

The Ethereum sale was uncapped, and ran for 42 days. The sale price was 2000 ETH for 1 BTC for the first 14 days, and then started increasing linearly, finishing at 1337 ETH for 1 BTC.
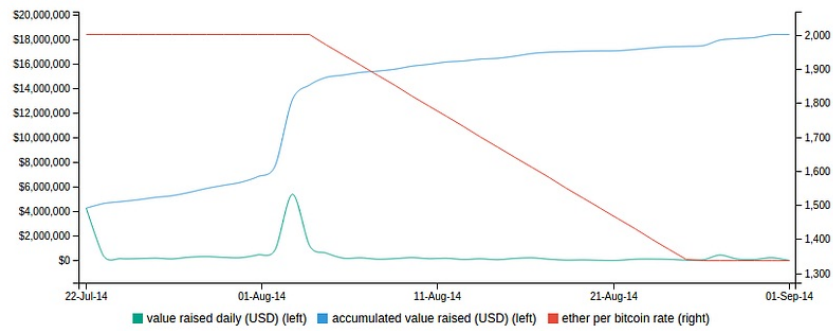
Nearly every uncapped sale is criticized for being "greedy" (a criticism I have significant reservations about, but we'll get back to this later), though there is also another more interesting criticism of these sales: they give participants *high uncertainty about the valuation* that they are buying at. To use a not-yet-started sale as a example, there are likely many people who would be willing to pay $10,000 for a pile of Bancor tokens if they knew for a fact that this pile represented 1% of all Bancor tokens in existence, but many of them would become quite apprehensive if they were buying a pile of, say, 5000 Bancor tokens, and they had no idea whether the total supply would be 50000, 500000 or 500 million.

In the Ethereum sale, buyers who really cared about predictability of valuation generally bought on the 14th day, reasoning that this was the last day of the full discount period and so on this day they had maximum predictability together with the full discount, but the pattern above is hardly economically optimal behavior; the equilibrium would be something like everyone buying in on the last hour of the 14th day, making a private tradeoff between certainty of valuation and taking the 1.5% hit (or, if certainty was really important, purchases could spill over into the 15th, 16th and later days). Hence, the model certainly has some rather weird economic properties that we would really like to avoid if there is a convenient way to do so.

BAT

Throughout 2016 and early 2017, the capped sale design was most popular. Capped sales have the property that it is very likely that interest is oversubscribed, and so there is a large incentive to getting in first. Initially, sales took a few hours to finish. However, soon the speed began to accelerate. First Blood made a lot of news by finishing their $5.5m sale in [two minutes](#) - *while [active denial-of-service attacks](#) on the Ethereum blockchain were taking place*. However, the apotheosis of this race-to-the-Nash-equilibrium did not come until the BAT sale last month, when a [$35m sale was completed within 30 seconds](#) due to the large amount of interest in the project.
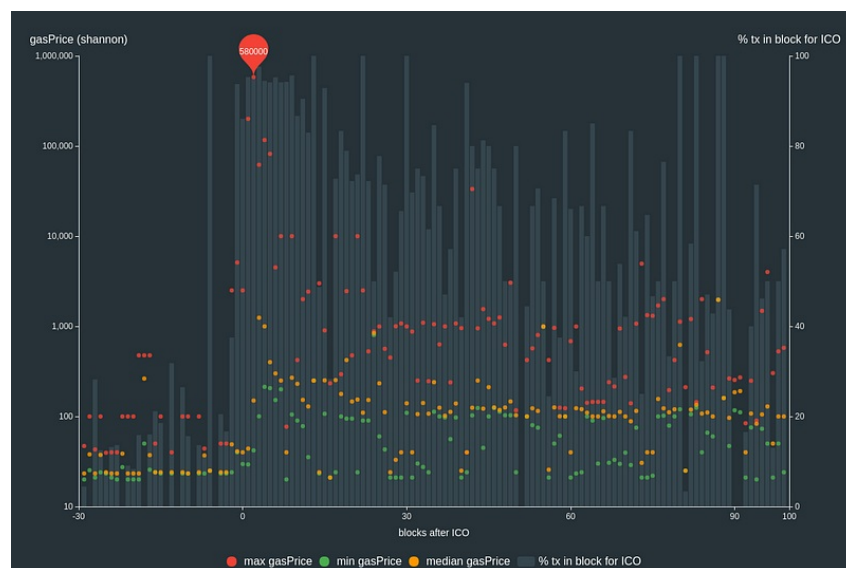
Not only did the sale finish within two blocks, but also:

- The total transaction fees paid were [70.15 ETH](#) (>$15,000), with the highest single fee being

~$6,600
- 185 purchases were successful, and over 10,000 failed
- The Ethereum blockchain's capacity was full for 3 hours after the sale started

Thus, we are starting to see capped sales approach their natural equilibrium: people trying to outbid each other's transaction fees, to the point where potentially millions of dollars of surplus would be burned into the hands of miners. And that's before the next stage starts: large mining pools butting into the start of the line and just buying up all of the tokens themselves before anyone else can.

Gnosis

The Gnosis sale attempted to alleviate these issues with a novel mechanism: the reverse dutch auction. The terms, in simplified form, are as follows. There was a capped sale, with a cap of $12.5 million USD. However, the portion of tokens that would actually be given to purchasers depended on how long the sale took to finish. If it finished on the first day, then only ~5% of tokens would be distributed among purchasers, and the rest held by the Gnosis team; if it finished on the second day, it would be ~10%, and so forth.

The purpose of this is to create a scheme where, if you buy at time T, then you are guaranteed to buy in at a valuation which is at most $\frac{1}{T}$.
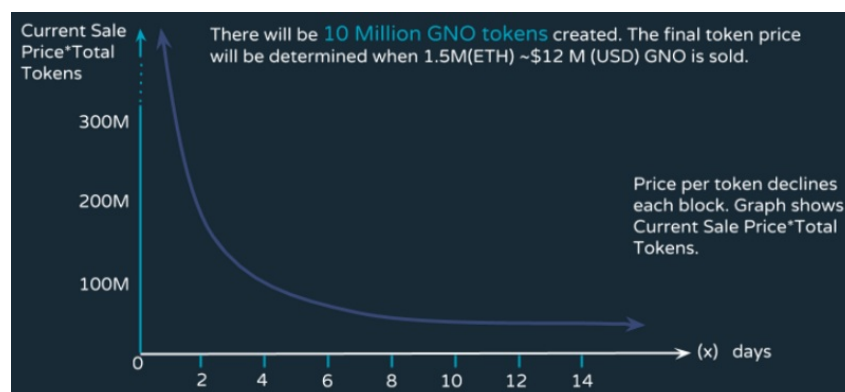
The goal is to create a mechanism where the optimal strategy is simple. First, you personally decide what is the highest valuation you would be willing to buy at (call it V). Then, when the sale starts, you don't buy in immediately; rather, you wait until the valuation drops to below that level, and then send your transaction.

There are two possible outcomes:

1. The sale closes before the valuation drops to below V. Then, you are happy because you stayed out of what you thought is a bad deal.
2. The sale closes after the valuation drops to below V. Then, you sent your transaction, and you are happy because you got into what you thought is a good deal.

However, many people predicted that because of "fear of missing out" (FOMO), many people would just "irrationally" buy in at the first day, without even looking at the valuation. And this is exactly what happened: the sale finished in a few hours, with the result that the sale reached its cap of $12.5 million when it was only selling about 5% of all tokens that would be in existence - an implied valuation of over $300 million.

All of this would of course be an excellent piece of confirming evidence for the narrative that markets are totally irrational, people don't think clearly before throwing in large quantities of money (and often, as a subtext, that the entire space needs to be somehow suppressed to prevent further exuberance) if it weren't for one inconvenient fact: **the traders who bought into the sale were right**.

Gnosis Charts

Even in ETH terms, despite the massive ETH price rise, the price of 1 GNO has increased from ~0.6 ETH to ~0.8 ETH.

What happened? A couple of weeks before the sale started, facing public criticism that if they end up holding the majority of the coins they would act like a central bank with the ability to heavily manipulate GNO prices, the Gnosis team agreed to hold 90% of the coins that were not sold for a year. From a trader's point of view, coins that are locked up for a long time are coins that cannot affect the market, and so in a short term analysis, might as well not exist. This is what initially propped up Steem to such a high valuation last year in July, as well as Zcash in the very early moments when the price of each coin was over $1,000.

Now, one year is not *that* long a time, and locking up coins for a year is nowhere close to the same thing as locking them up forever. However, the reasoning goes further. Even after the one year holding period expires, you can argue that it is in the Gnosis team's interest to only release the locked coins if they believe that doing so will make the price go up, and so if you trust the Gnosis team's judgement this means that they are going to do something *which is at least as good for the GNO price as simply locking up the coins forever*. Hence, in reality, the GNO sale was really much more like a capped sale with a cap of $12.5 million and a valuation of $37.5 million. And the traders who participated in the sale reacted exactly as they should have, leaving scores of internet commentators wondering what just happened.

There is certainly a weird bubbliness about crypto-assets, with various no-name assets attaining market caps of $1-100 million (including BitBean as of the time of this writing at $12m, PotCoin at $22m, PepeCash at $13m and SmileyCoin at $14.7m) just because. However, there's a strong case to be made that the participants *at the sale stage* are in many cases doing nothing wrong, at least for themselves; rather, traders who buy in sales are simply (correctly) predicting the existence of an ongoing bubble has been brewing since the start of 2015 (and arguably, since the start of 2010).

More importantly though, bubble behavior aside, there is another legitimate criticism of the Gnosis sale: despite their 1-year no-selling promise, eventually they will have access to the entirety of their coins, and they **will** to a limited extent be able to act like a central bank with the ability to heavily manipulate GNO prices, and traders will have to deal with all of the monetary policy uncertainty that that entails.

Specifying the problem

So what would a **good** token sale mechanism look like? One way that we can start off is by looking through the criticisms of existing sale models that we have seen and coming up with a list of desired properties.

Let's do that. Some natural properties include:

1. **Certainty of valuation** - if you participate in a sale, you should have certainty over at least a ceiling on the valuation (or, in other words, a floor on the percentage of all tokens you are getting).
2. **Certainty of participation** - if you try to participate in a sale, you should be able to generally count on succeeding.
3. **Capping the amount raised** - to avoid being perceived as greedy (or possibly to mitigate risk of regulatory attention), the sale should have a limit on the amount of money it is collecting.
4. **No central banking** - the token sale issuer should not be able to end up with an unexpectedly very large percentage of the tokens that would give them control over the market.
5. **Efficiency** - the sale should not lead to substantial economic inefficiencies or deadweight losses.

Sounds reasonable?

Well, here's the not-so-fun part.

- (1) and (2) cannot be fully satisfied simultaneously.
- At least without resorting to very clever tricks, (3), (4) and (5) cannot be satisfied simultaneously.

These can be cited as "the first token sale dilemma" and "the second token sale trilemma".

The proof for the first dilemma is easy: suppose you have a sale where you provide users with certainty of a $100 million valuation. Now, suppose that users try to throw $101 million into the sale. At least some will fail. The proof for the second trilemma is a simple supply-and-demand argument. If you satisfy (4), then you are selling all, or some fixed large percentage, of the tokens, and so the valuation you are selling at is proportional to the price you are selling at. If you satisfy (3), then you are putting a cap on the price. However, this implies the possibility that the equilibrium price at the quantity you are selling exceeds the price cap that you set, and so you get a shortage, which inevitably leads to either (i) the digital equivalent of standing in line for 4 hours at a very popular restaurant, or (ii) the digital equivalent of ticket scalping - both large deadweight losses, contradicting (5).

The first dilemma cannot be overcome; some valuation uncertainty or participation uncertainty is inescapable, though when the choice exists it seems better to try to choose participation uncertainty rather than valuation uncertainty. The closest that we can come is compromising on *full participation* to *guarantee* partial participation. This can be done with a proportional refund (eg. if $101 million buy in at a $100 million valuation, then everyone gets a 1% refund). We can also think of this mechanism as being an uncapped sale where part of the payment comes in the form of *locking up* capital rather than spending it; from this viewpoint, however, it becomes clear that the requirement to lock up capital is an efficiency loss, and so such a mechanism fails to satisfy (5). If ether holdings are not well-distributed then it arguably harms fairness by favoring wealthy stakeholders.

The second dilemma is difficult to overcome, and many attempts to overcome it can easily fail or backfire. For example, the Bancor sale is considering limiting the transaction gas price for purchases to 50 shannon (~12x the normal gasprice). However, this now means that the optimal strategy for a buyer is to set up a large number of accounts, and from each of those accounts send a transaction that triggers a contract, which then attempts to buy in (the indirection is there to make it impossible for the buyer to accidentally buy in more than they wanted, and to reduce capital requirements). The more accounts a buyer sets up, the more likely they are to get in. Hence, in equilibrium, this could lead to *even more* clogging of the Ethereum blockchain than a BAT-style sale, where at least the $6600 fees were spent on a single transaction and not an entire denial-of-service attack on the network. Furthermore, any kind of on-chain transaction spam contest severely harms fairness, because the cost of participating in the contest is constant, whereas the reward is proportional to how much money you have, and so the result disproportionately favors wealthy stakeholders.

Moving forward

There are three more clever things that you can do. First, you can do a reverse dutch auction just like Gnosis, but with one change: instead of holding the unsold tokens, put them toward some kind of public good. Simple examples include: (i) airdrop (ie. redistributing to all ETH holders), (ii) donating to the Ethereum Foundation, (iii) donating to Parity, Brainbot, Smartpool or other companies and individuals independently building infrastructure for the Ethereum space, or (iv) some combination of all three, possibly with the ratios somehow being voted on by the token buyers.

Second, you can keep the unsold tokens, but solve the "central banking" problem by committing to a fully automated plan for how they would be spent. The reasoning here is similar to that for why many economists are interested in rules-based monetary policy: even if a centralized entity has a large amount of control over a powerful resource, much of the political uncertainty that results can be mitigated if the entity credibly commits to following a set of programmatic rules for how they apply it. For example, the unsold tokens can be put into a market maker that is tasked with preserving the tokens' price stability.

Third, you can do a capped sale, where you limit the amount that can be bought by each person. Doing this effectively requires a KYC process, but the nice thing is a KYC entity can do this once, whitelisting users' addresses after they verify that the address represents a unique individual, and this can then be reused for every token sale, alongside other applications that can benefit from per-person sybil resistance like Akasha's quadratic voting. There is still deadweight loss (ie. inefficiency) here, because this will lead to individuals with no personal interest in tokens participating in sales because they know they will be able to quickly flip them on the market for a profit. However, this is

arguably not that bad: it creates a kind of [crypto universal basic income](#), and if behavioral economics assumptions like the [endowment effect](#) are even slightly true it will also succeed at the goal of ensuring widely distributed ownership.

Are single round sales even good?

Let us get back to the topic of "greed". I would claim that not many people are, in principle, opposed to the idea of development teams that are capable of spending $500 million to create a really great project getting $500 million. Rather, what people are opposed to is (i) the idea of completely new and untested development teams getting $50 million all at once, and (ii) even more importantly, the *time mismatch between developers' rewards and token buyers' interests*. In a single-round sale, the developers have only one chance to get money to build the project, and that is near the start of the development process. There is no feedback mechanism where teams are first given a small amount of money to prove themselves, and then given access to more and more capital over time as they prove themselves to be reliable and successful. During the sale, there is comparatively little information to filter between good development teams and bad ones, and once the sale is completed, the incentive to developers to keep working is relatively low compared to traditional companies. The "greed" isn't about getting lots of money, it's about getting lots of money without working hard to show you're capable of spending it wisely.

If we want to strike at the heart of this problem, how would we solve it? I would say the answer is simple: start moving to mechanisms other than single round sales.

I can offer several examples as inspiration:

- **[Angelshares](#)** - this project ran a sale in 2014 where it sold off a fixed percentage of all AGS every day for a period of several months. During each day, people could contribute an unlimited amount to the crowdsale, and the AGS allocation for that day would be split among all contributors. Basically, this is like having a hundred "micro-rounds" of uncapped sales over the course of most of a year; I would claim that the duration of the sales could be stretched even further.
- **[Mysterium](#)**, which held a little-noticed [micro-sale](#) six months before the big one.
- **[Bancor](#)**, which [recently agreed](#) to put all funds raised over a cap into a market maker which will maintain price stability along with maintaining a price floor of 0.01 ETH. These funds cannot be removed from the market maker for two years.

It seems hard to see the relationship between Bancor's strategy and solving time mismatch incentives, but an element of a solution is there. To see why, consider two scenarios. As a first case, suppose the sale raises $30 million, the cap is $10 million, but then after one year everyone agrees that the project is a flop. In this case, the price would try to drop below 0.01 ETH, and the market maker would lose all of its money trying to maintain the price floor, and so the team would only have $10 million to work with. As a second case, suppose the sale raises $30 million, the cap is $10 million, and after two years everyone is happy with the project. In this case, the market maker will not have been triggered, and the team would have access to the entire $30 million.

A related proposal is Vlad Zamfir's "[safe token sale mechanism](#)". The concept is a very broad one that could be parametrized in many ways, but one way to parametrize it is to sell coins at a price ceiling and then have a price floor slightly below that ceiling, and then allow the two to diverge over time, freeing up capital for development over time if the price maintains itself.

Arguably, none of the above three are sufficient; we want sales that are spread out over an even longer period of time, giving us much more time to see which development teams are the most worthwhile before giving them the bulk of their capital. But nevertheless, this seems like the most productive direction to explore in.

Coming out of the Dilemmas

From the above, it should hopefully be clear that while there is no way to counteract the dilemma and trilemma head on, there are ways to chip away at the edges by thinking outside the box and compromising on variables that are not apparent from a simplistic view of the problem. We can compromise on guarantee of participation slightly, mitigating the impact by using time as a third dimension: if you don't get in during round N, you can just wait until round $N+1$ which will be in a week and where the price probably will not be that different.

We can have a sale which is uncapped as a whole, but which consists of a variable number of periods, where the sale within each period is capped; this way teams would not be asking for very large amounts of money without proving their ability to handle smaller rounds first. We can sell small

portions of the token supply at a time, removing the political uncertainty that this entails by putting the remaining supply into a contract that continues to sell it automatically according to a prespecified formula.

Here are a few possible mechanisms that follow some of the spirit of the above ideas:

- Host a Gnosis-style reverse dutch auction with a low cap (say, $1 million). If the auction sells less than 100% of the token supply, automatically put the remaining funds into another auction two months later with a 30% higher cap. Repeat until the entire token supply is sold.
- Sell an unlimited number of tokens at a price of $\$X$ and put 90% of the proceeds into a smart contract that guarantees a price floor of $\$0.9 \cdot X$. Have the price ceiling go up hyperbolically toward infinity, and the price floor go down linearly toward zero, over a five-year period.
- Do the exact same thing AngelShares did, though stretch it out over 5 years instead of a few months.
- Host a Gnosis-style reverse dutch auction. If the auction sells less than 100% of the token supply, put the remaining funds into an automated market maker that attempts to ensure the token's price stability (note that if the price continues going up anyway, then the market maker would be selling tokens, and some of these earnings could be given to the development team).
- Immediately put all tokens into a market maker with parameters+variables X (minimum price), s (fraction of all tokens already sold), t (time since sale started), T (intended duration of sale, say 5 years), that sells tokens at a price of $\dfrac{k}{(\frac{t}{T} - s)}$ (this one is weird and may need to be economically studied more).

Note that there are other mechanisms that should be tried to solve other problems with token sales; for example, revenues going into a multisig of curators, which only hand out funds if milestones are being met, is one very interesting idea that should be done more. However, the design space is highly multidimensional, and there are a lot more things that could be tried.

Engineering Security Through Coordination Problems

2017 May 08

Recently, there was a small spat between the Core and Unlimited factions of the Bitcoin community, a spat which represents perhaps the fiftieth time the same theme was debated, but which is nonetheless interesting because of how it highlights a very subtle philosophical point about how blockchains work.

ViaBTC, a mining pool that favors Unlimited, tweeted "hashpower is law", a usual talking point for the Unlimited side, which believes that miners have, and should have, a very large role in the governance of Bitcoin, the usual argument for this being that miners are the one category of users that has a large and illiquid financial incentive in Bitcoin's success. Greg Maxwell (from the Core side) replied that "Bitcoin's security works precisely because hash power is NOT law".

The Core argument is that miners only have a limited role in the Bitcoin system, to secure the ordering of transactions, and they should NOT have the power to determine anything else, including block size limits and other block validity rules. These constraints are enforced by full nodes run by users - if miners start producing blocks according to a set of rules different than the rules that users' nodes enforce, then the users' nodes will simply reject the blocks, regardless of whether 10% or 60% or 99% of the hashpower is behind them. To this, Unlimited often replies with something like "if 90% of the hashpower is behind a new chain that increases the block limit, and the old chain with 10% hashpower is now ten times slower for five months until difficulty readjusts, would you *really* not update your client to accept the new chain?"

Many people often argue against the use of public blockchains for applications that involve real-world assets or anything with counterparty risk. The critiques are either total, saying that there is no point in implementing such use cases on public blockchains, or partial, saying that while there may be advantages to storing the *data* on a public chain, the *business logic* should be executed off chain.

The argument usually used is that in such applications, points of trust exist already - there is someone who owns the physical assets that back the on-chain permissioned assets, and that someone could always choose to run away with the assets or be compelled to freeze them by a government or bank, and so managing the digital representations of these assets on a blockchain is like paying for a reinforced steel door for one's house when the window is open. Instead, such systems should use private chains, or even traditional server-based solutions, perhaps adding bits and pieces of cryptography to improve auditability, and thereby save on the inefficiencies and costs of putting everything on a blockchain.

The arguments above are both flawed in their pure forms, and they are flawed in a similar way. While it is *theoretically possible* for miners to switch 99% of their hashpower to a chain with new rules (to make an example where this is uncontroversially bad, suppose that they are increasing the block reward), and even spawn-camp the old chain to make it permanently useless, and it is also theoretically possible for a centralized manager of an asset-backed currency to cease honoring one digital token, make a new digital token with the same balances as the old token except with one particular account's balance reduced to zero, and start honoring the new token, in practice *those things are both quite hard to do*.

In the first case, users will have to realize that something is wrong with the existing chain, agree that they should go to the new chain that the miners are now mining on, and download the software that accepts the new rules. In the second case, all clients and applications that depend on the original digital token will break, users will need to update their clients to switch to the new digital token, and smart contracts with no capacity to look to the outside world and see that they need to update will break entirely. In the middle of all this, opponents of the switch can create a fear-uncertainty-and-doubt campaign to try to convince people that maybe they shouldn't update their clients after all, or update their client to some *third* set of rules (eg. changing proof of work), and this makes implementing the switch even more difficult.

Hence, we can say that in both cases, even though there theoretically are centralized or quasi-centralized parties that could force a transition from state A to state B, where state B is disagreeable to users but preferable to the centralized parties, doing so requires **breaking through a hard coordination problem**. Coordination problems are everywhere in society and are often a bad thing - while it would be better for most people if the English language got rid of its highly complex and irregular spelling system and made a phonetic one, or if the United States switched to metric, or if we could immediately [drop all prices and wages by ten percent in the event of a recession](link), in practice this requires everyone to agree on the switch at the same time, and this is often very very hard.

With blockchain applications, however, we are doing something different: **we are using coordination problems to our advantage**, using the friction that coordination problems create as a bulwark against malfeasance by centralized actors. We can build systems that have property X, and we can guarantee that they will preserve property X to a high degree because changing the rules from X to not-X would require a whole bunch of people to agree to update their software at the same time. Even if there is an actor that could force the change, doing so would be hard. This is the kind of security that you gain from client-side validation of blockchain consensus rules.

Note that this kind of security relies on the decentralization of users specifically. Even if there is only one miner in the world, there is still a difference between a cryptocurrency mined by that miner and a PayPal-like centralized system. In the latter case, the operator can choose to arbitrarily change the rules, freeze people's money, offer bad service, jack up their fees or do a whole host of other things, and the coordination problems are in the operator's favor, as such systems have substantial network effects and so very many users would have to agree at the same time to switch to a better system. In the former case, client-side validation means that many attempts at mischief that the miner might want to engage in are by default rejected, and the coordination problem now works in the users' favor.

Note that the arguments above do NOT, *by themselves*, imply that it is a bad idea for miners to be the principal actors coordinating and deciding the block size (or in Ethereum's case, the gas limit). It may well be the case that, *in the specific case of the block size/gas limit*, "government by coordinated miners with aligned incentives" is the optimal approach for deciding this one particular policy parameter, perhaps because the risk of miners abusing their power is lower than the risk that any specific chosen hard limit will prove wildly inappropriate for market conditions a decade after the limit is set. However, there is nothing unreasonable about saying that government-by-miners is the best way to decide one policy parameter, and at the same saying that *for other parameters* (eg. block reward) we want to rely on client-side validation to ensure that miners are constrained. This is the essence of engineering decentralized instutitions: it is about strategically using coordination problems to ensure that systems continue to satisfy certain desired properties.

The arguments above also do not imply that it is always optimal to try to put everything onto a blockchain even for services that are trust-requiring. There generally are at least some gains to be made by running more business logic on a blockchain, but they are often much smaller than the losses to efficiency or privacy. And this ok; the blockchain is not the best tool for every task. What the arguments above *do* imply, though, is that if you are building a blockchain-based application that contains many centralized components out of necessity, then you can make substantial further gains in trust-minimization by giving users a way to access your application through a regular blockchain client (eg. in the case of Ethereum, this might be Mist, Parity, Metamask or Status), instead of getting them to use a web interface that you personally control.


Theoretically, the benefits of user-side validation are optimized if literally every user runs an independent "ideal full node" - a node that accepts all blocks that follow the protocol rules that everyone agreed to when creating the system, and rejects all blocks that do not. In practice, however, this involves asking every user to process every transaction run by everyone in the network, which is clearly untenable, especially keeping in mind the rapid growth of smartphone users in the developing world.

There are two ways out here. The first is that we can realize that while it is *optimal* from the point of view of the above arguments that everyone runs a full node, it is certainly not *required*. Arguably, any major blockchain running at full capacity will have already reached the point where it will not make sense for "the common people" to expend a fifth of their hard drive space to run a full node, and so the remaining users are hobbyists and businesses. As long as there is a fairly large number of them, and they come from diverse backgrounds, the coordination problem of getting these users to collude will still be very hard.

Second, we can rely on **strong light client technology**.

There are two levels of "light clients" that are generally possible in blockchain systems. The first, weaker, kind of light client simply convinces the user, with some degree of economic assurance, that they are on the chain that is supported by the majority of the network. This can be done much more cheaply than verifying the entire chain, as all clients need to do is in proof of work schemes verify nonces or in proof stake schemes verify signed certificates that state "either the root hash of the state is what I say it is, or you can publish this certificate into the main chain to delete a large amount of my money". Once the light client verifies a root hash, they can use Merkle trees to verify any specific piece of data that they might want to verify.
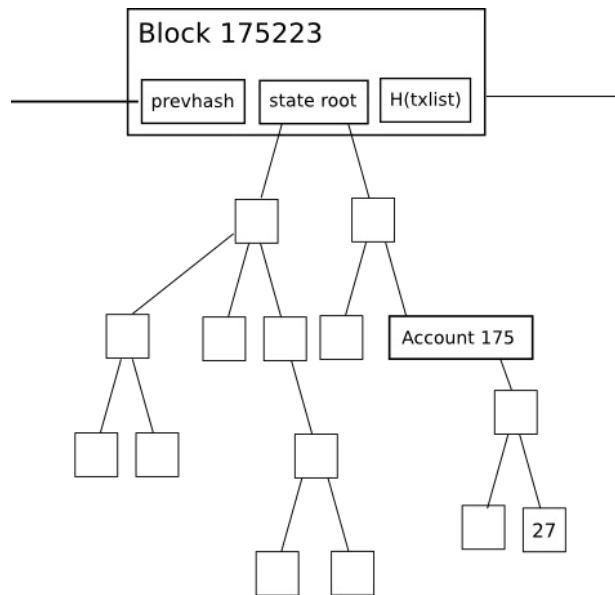
Look, it's a Merkle tree!

The second level is a "nearly fully verifying" light client. This kind of client doesn't just try to follow the chain that the majority follows; rather, it also tries to follow only chains that follow all the rules. This is done by a combination of strategies; the simplest to explain is that a light client can work together with specialized nodes (credit to Gavin Wood for coming up with the name "fishermen") whose purpose is to look for blocks that are invalid and generate "fraud proofs", short messages that essentially say "Look! This block has a flaw over here!". Light clients can then verify that specific part of a block and check if it's actually invalid.

If a block is found to be invalid, it is discarded; if a light client does not hear any fraud proofs for a given block for a few minutes, then it assumes that the block is probably legitimate. There's a [bit more complexity](#) involved in handling the case where the problem is not data that is *bad*, but rather data that is *missing*, but in general it is possible to get quite close to catching all possible ways that miners or validators can violate the rules of the protocol.

Note that in order for a light client to be able to efficiently validate a set of application rules, those rules must be executed inside of consensus - that is, they must be either part of the protocol or part of a mechanism executing inside the protocol (like a smart contract). This is a key argument in favor of using the blockchain for both data storage and business logic execution, as opposed to just data storage.

These light client techniques are imperfect, in that they do rely on assumptions about network connectivity and the number of other light clients and fishermen that are in the network. But it is actually not crucial for them to work 100% of the time for 100% of validators. Rather, all that we want is to create a situation where any attempt by a hostile cartel of miners/validators to push invalid blocks without user consent will cause a large amount of headaches for lots of people and ultimately require everyone to update their software if they want to continue to synchronize with the invalid chain. As long as this is satisfied, we have achieved the goal of security through coordination frictions.

Hard Forks, Soft Forks, Defaults and Coercion

2017 Mar 14

One of the important arguments in the blockchain space is that of whether hard forks or soft forks are the preferred protocol upgrade mechanism. The basic difference between the two is that soft forks change the rules of a protocol by *strictly reducing* the set of transactions that is valid, so nodes following the old rules will still get on the new chain (provided that the majority of miners/validators implements the fork), whereas hard forks allow previously invalid transactions and blocks to become valid, so clients must upgrade their clients in order to stay on the hard-forked chain. There are also two sub-types of hard forks: *strictly expanding* hard forks, which strictly expand the set of transactions that is valid, and so effectively the old rules are a soft fork with respect to the new rules, and *bilateral* hard forks, where the two rulesets are incompatible both ways.

Here is a Venn diagram to illustrate the fork types:
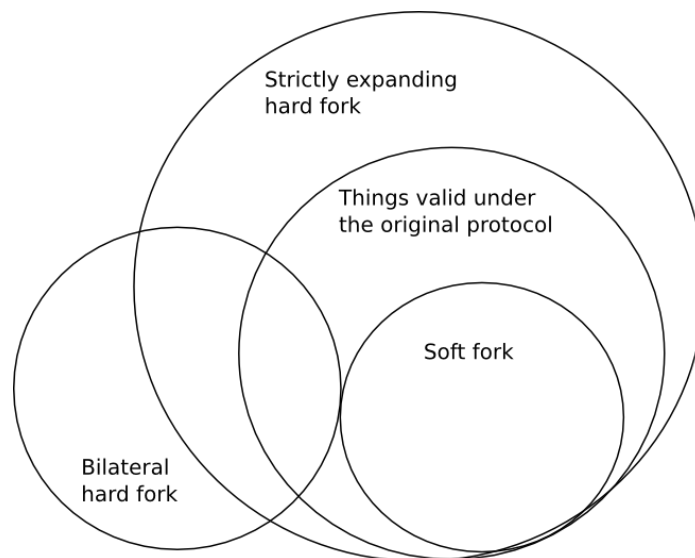
The benefits commonly cited for the two are as follows.

- Hard forks allow the developers much more flexibility in making the protocol upgrade, as they do not have to take care to make sure that the new rules "fit into" the old rules
- Soft forks are more convenient for users, as users do not need to upgrade to stay on the chain
- Soft forks are less likely to lead to a chain split
- Soft forks only really require consent from miners/validators (as even if users still use the old rules, if the nodes making the chain use the new rules then only things valid under the new rules will get into the chain in any case); hard forks require *opt-in* consent from users

Aside from this, one major criticism often given for hard forks is that hard forks are "coercive". The kind of coercion implied here is not physical force; rather, it's *coercion through network effect*. That is, if the network changes rules from A to B, then even if you personally like A, if most other users like B and switch to B then you have to switch to B despite your personal disapproval of the change in order to be on the same network as everyone else.

Proponents of hard forks are often derided as trying to effect a "hostile take over" of a network, and "force" users to go along with them. Additionally, the risk of chain splits is often used to bill hard forks as "unsafe".

It is my personal viewpoint that these criticisms are wrong, and furthermore in many cases completely backwards. This viewpoint is not specific to Ethereum, or Bitcoin, or any other blockchain; it arises out of general properties of these systems, and is applicable to any of them. Furthermore, the arguments below only apply to controversial changes, where a large portion of at least one constituency (miners/validators and users) disapprove of them; if a change is non-contentious, then it can generally be done safely no matter what the format of the fork is.

First of all, let us discuss the question of coercion. Hard forks and soft forks both change the protocol in ways that some users may not like; *any* protocol change will do this if it has less than exactly 100% support. Furthermore, it is almost inevitable that at least *some* of the dissenters, in any scenario, value the network effect of sticking with the larger group more than they value their own preferences regarding the protocol rules. Hence, both fork types are coercive, in the network-effect sense of the word.

However, there is an essential difference between hard forks and soft forks: *hard forks are opt-in, whereas soft forks allow users no "opting" at all*. In order for a user to join a hard forked chain, they must personally install the software package that implements the fork rules, and the set of users that disagrees with a rule change even more strongly than they value network effects can theoretically simply stay on the old chain - and, practically speaking, such an event [has already happened](#).

This is true in the case of both strictly expanding hard forks and bilateral hard forks. In the case of soft forks, however, *if the fork succeeds the unforked chain does not exist*. Hence, **soft forks clearly institutionally favor coercion over secession, whereas hard forks have the opposite bias**. My own moral views lead me to favor secession over coercion, though others may differ (the most common argument raised is that network effects are really really important and it is essential that "[one coin rule them all](#)", though more moderate versions of this also exist).

If I had to guess why, despite these arguments, soft forks are often billed as "less coercive" than hard forks, I would say that it is because it feels like a hard fork "forces" the user into installing a software update, whereas with a soft fork users do not "have" to do anything at all. However, this intuition is misguided: what matters is not whether or not individual users have to perform the simple bureaucratic step of clicking a "download" button, but rather whether or not the user is *coerced into accepting a change in protocol rules* that they would rather not accept. And by this metric, as mentioned above, both kinds of forks are ultimately coercive, and it is hard forks that come out as being somewhat better at preserving user freedom.

Now, let's look at highly controversial forks, particularly forks where miner/validator preferences and user preferences conflict. There are three cases here: (i) bilateral hard forks, (ii) strictly expanding hard forks, and (iii) so-called "user-activated soft forks" (UASF). A fourth category is where miners activate a soft fork *without user consent*; we will get to this later.

First, bilateral hard forks. In the best case, the situation is simple. The two coins trade on the market, and traders decide the relative value of the two. From the ETC/ETH case, we have overwhelming evidence that miners are overwhelmingly likely to simply assign their hashrate to coins based on the ratio of prices in order to maximize their profit, regardless of their own ideological views.
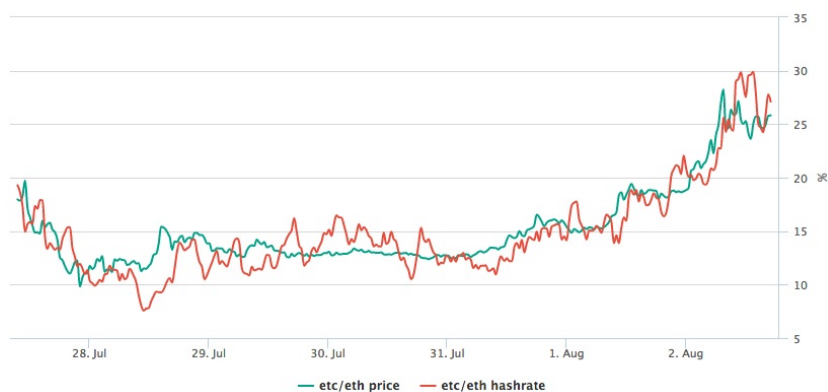
Even if some miners profess ideological preferences toward one side or the other, it is

overwhemingly likely that there will be enough miners that are willing to arbitrage any mismatch between price ratio and hashpower ratio, and bring the two into alignment. If a cartel of miners tries to form to not mine on one chain, there are overwheming incentives to defect.

There are two edge cases here. The first is the possibilty that, because of an inefficient difficulty adjustment algorithm, the value of mining the coin goes down becase price drops but difficulty does not go down to compensate, making mining very unprofitable, and there are no miners willing to mine at a loss to keep pushing the chain forward until its difficulty comes back into balance. This was not the case with Ethereum, but may well [be the case with Bitcoin](). Hence, the minority chain may well simply never get off the ground, and so it will die. Note that the normative question of *whether or not this is a good thing* depends on your views on coercion versus secession; as you can imagine from what I wrote above I personally believe that such minority-chain-hostile difficulty adjustment algorithms are bad.

The second edge case is that if the disparity is very large, the large chain can 51% attack the smaller chain. Even in the case of an ETH/ETC split with a 10:1 ratio, this has not happened; so it is certainly not a given. However, it is always a possibility if miners on the dominant chain prefer coercion to allowing secession and act on these values.

Next, let's look at strictly expanding hard forks. In an SEHF, there is the property that the non-forked chain is valid under the forked rules, and so if the fork has a lower price than the non-forked chain, it will have less hashpower than the non-forked chain, and so the non-forked chain will end up being accepted as the longest chain *by both original-client and forked-client rules* - and so the forked chain "[will be annihilated]()".

There is an argument that there is thus a strong inherent bias against such a fork succeeding, as the possibility that the forked chain will get annihilated will be baked into the price, pushing the price lower, making it even more likely that the chain will be annihilated... This argument to me seems strong, and so it is a very good reason to make *any* contentious hard fork bilateral rather than strictly expanding.

Bitcoin Unlimited developers suggest dealing with this problem by [making the hard fork bilateral manually]() after it happens, but a better choice would be to make the bilaterality built-in; for example, in the bitcoin case, one can add a rule to ban some unused opcode, and then make a transaction containing that opcode on the non-forked chain, so that under the forked rules the non-forked chain will from then on be considered forever invalid. In the Ethereum case, because of various details about how state calculation works, nearly all hard forks are bilateral almost automatically. Other chains may have different properties depending on their architecture.

The last type of fork that was mentioned above is the user-activated soft fork. In a UASF, users turn on the soft fork rules without bothering to get consensus from miners; miners are expected to simply fall in line out of economic interest. If many users do not go along with the UASF, then there will be a coin split, and this will lead to a scenario identical to the strictly expanding hard fork, except - and this is the really clever and devious part of the concept - *the same "risk of annihilation" pressure that strongly disfavors the forked chain in a strictly expanding hard fork instead strongly favors the forked chain in a UASF*. Even though a UASF is opt-in, it uses economic asymmetry in order to bias itself toward success (though the bias is not absolute; if a UASF is decidedly unpopular then it will not succeed and will simply lead to a chain split).

However, UASFs are a dangerous game. For example, let us suppose that the developers of a project want to make a UASF patch that converts an unused opcode that previously accepted all transactions into an opcode that only accepts transactions that comply with the rules of some cool new feature, though one that is politically or technically controversial and miners dislike. Miners have a clever and devious way to fight back: *they can unilaterally implement a miner-activated soft fork that makes all transactions using the feature created by the soft fork always fail*.

Now, we have three rulesets:

1. The original rules where opcode X is always valid.
2. The rules where opcode X is only valid if the rest of the transaction complies with the new rules
3. The rules where opcode X is always invalid.

Note that (2) is a soft-fork with respect to (1), and (3) is a soft-fork with respect to (2). Now, there is strong economic pressure in favor of (3), and so the soft-fork fails to accomplish its objective.

The conclusion is this. Soft forks are a dangerous game, and they become even more dangerous if they are contentious and miners start fighting back. Strictly expanding hard forks are also a dangerous game. Miner-activated soft forks are coercive; user-activated soft forks are less coercive, though still quite coercive because of the economic pressure, and they also have their dangers. If you really want to make a contentious change, and have decided that the high social costs of doing so are worth it, just do a clean bilateral hard fork, spend some time to add some proper replay protection, and let the market sort it out.

A Note On Charity Through Marginal Price Discrimination

2017 Mar 11

Updated 2018-07-28. See end note.

The following is an interesting idea that I had two years ago that I personally believe has promise and could be easily implemented in the context of a blockchain ecosystem, though if desired it could certainly also be implemented with more traditional technologies (blockchains would help get the scheme network effects by putting the core logic on a more neutral platform).

Suppose that you are a restaurant selling sandwiches, and you ordinarily sell sandwiches for $7.50. Why did you choose to sell them for $7.50, and not $7.75 or $7.25? It clearly can't be the case that the cost of production is exactly $7.49999, as in that case you would be making no profit, and would not be able to cover fixed costs; hence, in most normal situations you would still be able to make *some* profit if you sold at $7.25 or $7.75, though less. Why less at $7.25? Because the price is lower. Why less at $7.75? Because you get fewer customers. It just so happens that $7.50 is the point at which the balance between those two factors is optimal for you.

| Price	Demand	Profit (demand * (price - $6 production cost))
6.8	6600	5280
6.9	6300	5670
7	6000	6000
7.1	5700	6270
7.2	5400	6480
7.3	5100	6630
7.4	4800	6720
7.5	4500	6750
7.6	4200	6720
7.7	3900	6630
7.8	3600	6480
7.9	3300	6270
8	3000	6000
8.1	2700	5670
8.2	2400	5280

Profit/price relationship

Notice one consequence of this: if you make a *slight* distortion to the optimal price, then even compared to the magnitude of the distortion the losses that you face are minimal. If you raise prices by 1%, from $7.50 to $7.575, then your profit declines from $6750 to $6733.12-a tiny 0.25% reduction. And that's *profit*-if you had instead donated 1% of the price of each sandwich, it would have reduced your profit by 5%. The smaller the distortion the more favorable the ratio: raising prices by 0.2% only cuts your profits down by 0.01%.

Now, you could argue that stores are not perfectly rational, and not perfectly informed, and so they may not *actually* be charging at optimal prices, all factors considered. However, if you don't know what direction the deviation is in for any given store, then even still, *in expectation*, the scheme works the same way-except instead of losing $17 it's more like flipping a coin where half the time you gain $50 and half the time you lose $84. Furthermore, in the more complex scheme that we will describe later, we'll be adjusting prices in both directions simultaneously, and so there will not even be any extra risk - no matter how correct or incorrect the original price was, the scheme will give you a predictable small net loss.

Also, the above example was one where marginal costs are high, and customers are picky about prices-in the above model, charging $9 would have netted you no customers at all. In a situation where marginal costs are much lower, and customers are less price-sensitive, the losses from raising or lowering prices would be even lower.

So what is the point of all this? Well, suppose that our sandwich shop changes its policy: it sells sandwiches for $7.55 to the general public, but lowers the prices to $7.35 for people who volunteered in some charity that maintains some local park (say, this is 25% of the population). The store's new profit is $(\$6682.5 \cdot 0.25+\$6742.5 \cdot 0.75=\$6727.5)$ (that's a $22.5 loss), but the result is that you are now paying all 4500 of your customers 20 cents each to volunteer at that charity-an incentive size of $900 (if you just count the customers who actually do volunteer, $225). So the store loses a bit, but gets a huge amount of leverage, de-facto contributing at least $225 depending on how you measure it for a cost of only $22.5.
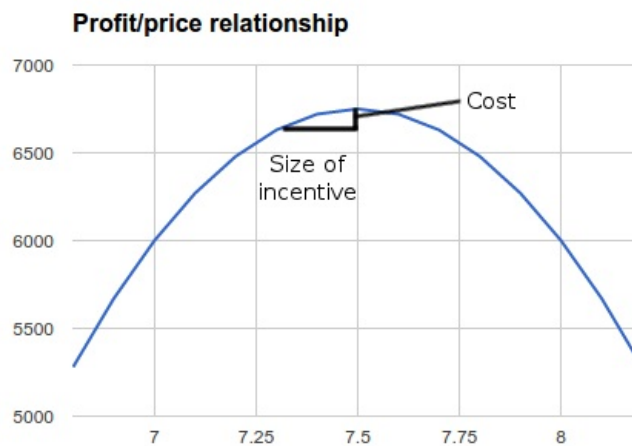
Now, what we can start to do is build up an ecosystem of "stickers", which are non-transferable digital "tokens" that organizations hand out to people who they think are contributing to worthy causes. Tokens could be organized by category (eg. poverty relief, science research, environmental, local community projects, open source software development, writing good blogs), and merchants would be free to charge marginally lower prices to holders of the tokens that represent whatever causes they personally approve of.

The next stage is to make the scheme recursive - being or working for a merchant that offers lower prices to holders of green stickers is itself enough to merit you a green sticker, albeit one that is of lower potency and gives you a lower discount. This way, if an entire community approves of a particular cause, it may actually be profit-maximizing to start offering discounts for the associated sticker, and so economic and social pressure will maintain a certain level of spending and participation toward the cause in a stable equilibrium.

As far as implementation goes, this requires:

- A standard for stickers, including wallets where people can hold stickers
- Payment systems that have support for charging lower prices to sticker holders included
- At least a few sticker-issuing organizations (the lowest overhead is likely to be issuing stickers for charity donations, and for easily verifiable online content, eg. open source software and blogs)

So this is something that can certainly be bootstrapped within a small community and user base and then let to grow over time.

Update 2017.03.14: here is an economic model/simulation showing the above implemented as a

Python script.

Update 2018.07.28: after discussions with others (Glen Weyl and several Reddit commenters), I realized a few extra things about this mechanism, some encouraging and some worrying:

- The above mechanism could be used not just by charities, but also by centralized corporate actors. For example, a large corporation could offer a bribe of $40 to any store that offers the 20-cent discount to customers of its products, gaining additional revenue much higher than $40. So it's empowering but potentially dangerous in the wrong hands... (I have not researched it but I'm sure this kind of technique is used in various kinds of loyalty programs already)
- The above mechanism has the property that a merchant can "donate" $\$x$ to charity at a cost of $\$x^{2}$ (note: $x^{2}<x$) at the scales we're talking about here). This gives it a structure that's economically optimal in certain ways (see [quadratic voting](#)), as a merchant that feels twice as strongly about some public good will be inclined to offer twice as large a subsidy, whereas most other social choice mechanisms tend to either undervalue (as in traditional voting) or overvalue (as in buying policies via lobbying) stronger vs weaker preferences.

[Mirror] Zk-SNARKs: Under the Hood

2017 Feb 01

This is a mirror of the post at https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6

This is the third part of a series of articles explaining how the technology behind zk-SNARKs works; the previous articles on quadratic arithmetic programs and elliptic curve pairings are required reading, and this article will assume knowledge of both concepts. Basic knowledge of what zk-SNARKs are and what they do is also assumed. See also Christian Reitwiessner's article here for another technical introduction.

In the previous articles, we introduced the quadratic arithmetic program, a way of representing any computational problem with a polynomial equation that is much more amenable to various forms of mathematical trickery. We also introduced elliptic curve pairings, which allow a very limited form of one-way homomorphic encryption that lets you do equality checking. Now, we are going to start from where we left off, and use elliptic curve pairings, together with a few other mathematical tricks, in order to allow a prover to prove that they know a solution for a particular QAP without revealing anything else about the actual solution.

This article will focus on the Pinocchio protocol by Parno, Gentry, Howell and Raykova from 2013 (often called PGHR13); there are a few variations on the basic mechanism, so a zk-SNARK scheme implemented in practice may work slightly differently, but the basic principles will in general remain the same.

To start off, let us go into the key cryptographic assumption underlying the security of the mechanism that we are going to use: the *knowledge-of-exponent* assumption.

KEA1: For any adversary **A** that takes input q, g, g^a and returns (C, Y) with $Y = C^a$, there exists an "extractor" $\bar{\textbf{A}}$, which given the same inputs as **A** returns c such that $g^c = C$.

Basically, if you get a pair of points P and Q, where $P \cdot k = Q$, and you get a point C, then it is not possible to come up with $C \cdot k$ unless C is "derived" from P in some way that you know. This may seem intuitively obvious, but this assumption actually cannot be derived from any other assumption (eg. discrete log hardness) that we usually use when proving security of elliptic curve-based protocols, and so zk-SNARKs do in fact rest on a somewhat shakier foundation than elliptic curve cryptography more generally — although it's still sturdy enough that most cryptographers are okay with it.

Now, let's go into how this can be used. Supposed that a pair of points (P, Q) falls from the sky, where $P \cdot k = Q$, but nobody knows what the value of k is. Now, suppose that I come up with a pair of points (R, S) where $R \cdot k = S$. Then, the KoE assumption implies that the only way I could have made that pair of points was by taking P and Q, and multiplying both by some factor r *that I personally know*. Note also that thanks to the magic of elliptic curve pairings, checking that $R = k \cdot S$ *doesn't actually require knowing k* - instead, you can simply check whether or not $e(R, Q) = e(P, S)$.

Let's do something more interesting. Suppose that we have ten pairs of points fall from the sky: $((P_1, Q_1), (P_2, Q_2)... (P_{10}, Q_{10}))$. In all cases, $P_i \cdot k = Q_i$. Suppose that I then provide you with a pair of points $((R, S))$ where $R \cdot k = S$. What do you know now? *You know that R is some linear combination $P_1 \cdot i_1 + P_2 \cdot i_2 + ... + P_{10} \cdot i_{10}$, where I know the coefficients $i_1, i_2 ... i_{10}$.* That is, the only way to arrive at such a pair of points $((R, S))$ is to take some multiples of $P_1, P_2 ... P_{10}$ and add them together, and make the same calculation with $Q_1, Q_2 ... Q_{10}$.

Note that, given any specific set of $P_1...P_{10}$ points that you might want to check linear combinations for, you can't actually create the accompanying $Q_1...Q_{10}$ points without knowing what k is, and if you do know what k is then you can create a pair $((R, S))$ where $R \cdot k = S$ for whatever R you want, without bothering to create a linear combination. Hence, for this to work it's absolutely imperative that whoever creates those points is trustworthy and actually deletes k once they created the ten points. **This is where the concept of a "trusted**

setup" comes from.

Remember that the solution to a QAP is a set of polynomials (A, B, C) such that $A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$, where:

- A is a linear combination of a set of polynomials $\{A_1...A_m\}$
- B is the linear combination of $\{B_1...B_m\}$ with the same coefficients
- C is a linear combination of $\{C_1...C_m\}$ with the same coefficients

The sets $\{A_1...A_m\}, \{B_1...B_m\}$ and $\{C_1...C_m\}$ and the polynomial Z are part of the problem statement.

However, in most real-world cases, A, B and C are extremely large; for something with many thousands of circuit gates like a hash function, the polynomials (and the factors for the linear combinations) may have many thousands of terms. Hence, instead of having the prover provide the linear combinations directly, we are going to use the trick that we introduced above to have the prover prove that they are providing something which is a linear combination, but without revealing anything else.

You might have noticed that the trick above works on elliptic curve points, not polynomials. Hence, what actually happens is that we add the following values to the trusted setup:

- $G \cdot A_1(t), G \cdot A_1(t) \cdot k_a$
- $G \cdot A_2(t), G \cdot A_2(t) \cdot k_a$
- ...
- $G \cdot B_1(t), G \cdot B_1(t) \cdot k_b$
- $G \cdot B_2(t), G \cdot B_2(t) \cdot k_b$
- ...
- $G \cdot C_1(t), G \cdot C_1(t) \cdot k_c$
- $G \cdot C_2(t), G \cdot C_2(t) \cdot k_c$
- ...

You can think of t as a "secret point" at which the polynomial is evaluated. G is a "generator" (some random elliptic curve point that is specified as part of the protocol) and t, k_a, k_b and k_c are "toxic waste", numbers that absolutely must be deleted at all costs, or else whoever has them will be able to make fake proofs. Now, if someone gives you a pair of points P, Q such that $P \cdot k_a = Q$ (reminder: we don't need k_a to check this, as we can do a pairing check), then you know that what they are giving you is a linear combination of A_i polynomials evaluated at t.

Hence, so far the prover must give:

- $\pi_a = G \cdot A(t), \pi'_a = G \cdot A(t) \cdot k_a$
- $\pi_b = G \cdot B(t), \pi'_b = G \cdot B(t) \cdot k_b$
- $\pi_c = G \cdot C(t), \pi'_c = G \cdot C(t) \cdot k_c$

Note that the prover doesn't actually need to know (and shouldn't know!) t, k_a, k_b or k_c to compute these values; rather, the prover should be able to compute these values just from the points that we're adding to the trusted setup.

The next step is to make sure that all three linear combinations have the same coefficients. This we can do by adding another set of values to the trusted setup: $G \cdot (A_i(t) + B_i(t) + C_i(t)) \cdot b$, where b is another number that should be considered "toxic waste" and discarded as soon as the trusted setup is completed. We can then have the prover create a linear combination with these values with the same coefficients, and use the same pairing trick as above to verify that this value matches up with the provided $A + B + C$.

Finally, we need to prove that $A \cdot B - C = H \cdot Z$. We do this once again with a pairing

check:

$e(\pi_a, \pi_b) / e(\pi_c, G) ?= e(\pi_h, G \cdot Z(t))$

Where $\pi_h = G \cdot H(t)$. If the connection between this equation and $A \cdot B - C = H \cdot Z$ does not make sense to you, go back and read the [article on pairings](#).

We saw above how to convert A, B and C into elliptic curve points; G is just the generator (ie. the elliptic curve point equivalent of the number one). We can add $G \cdot Z(t)$ to the trusted setup. H is harder; H is just a polynomial, and we predict very little ahead of time about what its coefficients will be for each individual QAP solution. Hence, we need to add yet more data to the trusted setup; specifically the sequence:

$G, G \cdot t, G \cdot t^2, G \cdot t^3, G \cdot t^4 ...$.

In the Zcash trusted setup, the sequence here goes up to about 2 million; this is how many powers of t you need to make sure that you will always be able to compute $H(t)$, at least for the specific QAP instance that they care about. And with that, the prover can provide all of the information for the verifier to make the final check.

There is one more detail that we need to discuss. Most of the time we don't just want to prove in the abstract that some solution exists for some specific problem; rather, we want to prove either the correctness of some specific solution (eg. proving that if you take the word "cow" and SHA3 hash it a million times, the final result starts with 0x73064fe5), or that a solution exists if you restrict some of the parameters. For example, in a cryptocurrency instantiation where transaction amounts and account balances are encrypted, you want to prove that you know some decryption key k such that:

1. `decrypt(old_balance, k) >= decrypt(tx_value, k)`

2. `decrypt(old_balance, k) - decrypt(tx_value, k) = decrypt(new_balance, k)`

The encrypted `old_balance`, `tx_value` and `new_balance` should be specified publicly, as those are the specific values that we are looking to verify at that particular time; only the decryption key should be hidden. Some slight modifications to the protocol are needed to create a "custom verification key" that corresponds to some specific restriction on the inputs.

Now, let's step back a bit. First of all, here's the verification algorithm in its entirety, courtesy of ben [Sasson, Tromer, Virza and Chiesa](#):

(c) Verifier V

- INPUTS: verification key vk, input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^{n} x_i \mathsf{vk}_{\mathsf{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for A, B, C:
$$e(\pi_A, \mathsf{vk}_A) = e(\pi'_A, \mathcal{P}_2), \, e(\mathsf{vk}_B, \pi_B) = e(\pi'_B, \mathcal{P}_2),$$
$$e(\pi_C, \mathsf{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$
3. Check same coefficients were used:
$$e(\pi_K, \mathsf{vk}_\gamma) = e(\mathsf{vk}_{\vec{x}} + \pi_A + \pi_C, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_B) \ .$$
4. Check QAP divisibility:
$$e(\mathsf{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \mathsf{vk}_Z) \cdot e(\pi_C, \mathcal{P}_2) \ .$$
5. Accept if and only if all the above checks succeeded.

The first line deals with parametrization; essentially, you can think of its function as being to create a "custom verification key" *for the specific instance of the problem* where some of the arguments are specified. The second line is the linear combination check for A, B and C; the third line is the check that the linear combinations have the same coefficients, and the fourth line is the product check $A \cdot B - C = H \cdot Z$.

Altogether, the verification process is a few elliptic curve multiplications (one for each "public" input variable), and five pairing checks, one of which includes an additional pairing multiplication. The proof contains eight elliptic curve points: a pair of points each for $A(t), B(t)$ and $C(t)$, a point π_k for $b \cdot (A(t) + B(t) + C(t))$, and a point π_h for $H(t)$. Seven of these points are on the F_p curve (32 bytes each, as you can compress the y coordinate to a single bit), and in the Zcash implementation one point (π_b) is on the twisted curve in F_{p^2} (64 bytes), so the total size of the proof is ~288 bytes.

The two computationally hardest parts of creating a proof are:

- Dividing $(A \cdot B - C) / Z$ to get H (algorithms based on the [Fast Fourier transform](#) can do this in sub-quadratic time, but it's still quite computationally intensive)

- Making the elliptic curve multiplications and additions to create the $A(t), B(t), C(t)$ and $H(t)$ values and their corresponding pairs

The basic reason why creating a proof is so hard is the fact that what was a single binary logic gate in the original computation turns into an operation that must be cryptographically processed through elliptic curve operations if we are making a zero-knowledge proof out of it. This fact, together with the superlinearity of fast Fourier transforms, means that proof creation takes ~20–40 seconds for a Zcash transaction.

Another very important question is: can we try to make the trusted setup a little... less trust-demanding? Unfortunately we can't make it completely trustless; the KoE assumption itself precludes making independent pairs $(P_i, P_i \cdot k)$ without knowing what k is. However, we can increase security greatly by using N-of-N multiparty computation - that is, constructing the trusted setup between N parties in such a way that as long as at least one of the participants deleted their toxic waste then you're okay.

To get a bit of a feel for how you would do this, here's a simple algorithm for taking an existing set $(G, G \cdot t, G \cdot t^2, G \cdot t^3...)$, and "adding in" your own secret so that you need both your secret and the previous secret (or previous set of secrets) to cheat.

The output set is simply:

$(G, (G \cdot t) \cdot s, (G \cdot t^2) \cdot s^2, (G \cdot t^3) \cdot s^3...)$

Note that you can produce this set knowing only the original set and s, and the new set functions in the same way as the old set, except now using $t \cdot s$ as the "toxic waste" instead of t. As long as you and the person (or people) who created the previous set do not both fail to delete your toxic waste and later collude, the set is "safe".

Doing this for the complete trusted setup is quite a bit harder, as there are several values involved, and the algorithm has to be done between the parties in several rounds. It's an area of active research to see if the multi-party computation algorithm can be simplified further and made to require fewer rounds or made more parallelizable, as the more you can do that the more parties it becomes feasible to include into the trusted setup procedure. It's reasonable to see why a trusted setup between six participants who all know and work with each other might make some people uncomfortable, but a trusted setup with thousands of participants would be nearly indistinguishable from no trust at all - and if you're really paranoid, you can get in and participate in the setup procedure yourself, and be sure that you personally deleted your value.

Another area of active research is the use of other approaches that do not use pairings and the same trusted setup paradigm to achieve the same goal; see [Eli ben Sasson's recent presentation](#) for one alternative (though be warned, it's at least as mathematically complicated as SNARKs are!)

Special thanks to Ariel Gabizon and Christian Reitwiessner for reviewing.

[Mirror] Exploring Elliptic Curve Pairings

2017 Jan 14

*This is a mirror of the post at https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627*

Trigger warning: math.

One of the key cryptographic primitives behind various constructions, including deterministic threshold signatures, zk-SNARKs and other simpler forms of zero-knowledge proofs is the elliptic curve pairing. Elliptic curve pairings (or "bilinear maps") are a recent addition to a 30-year-long history of using elliptic curves for cryptographic applications including encryption and digital signatures; pairings introduce a form of "encrypted multiplication", greatly expanding what elliptic curve-based protocols can do. The purpose of this article will be to go into elliptic curve pairings in detail, and explain a general outline of how they work.

You're not expected to understand everything here the first time you read it, or even the tenth time; this stuff is genuinely hard. But hopefully this article will give you at least a bit of an idea as to what is going on under the hood.

Elliptic curves themselves are very much a nontrivial topic to understand, and this article will generally assume that you know how they work; if you do not, I recommend this article here as a primer: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/. As a quick summary, elliptic curve cryptography involves mathematical objects called "points" (these are literal two-dimensional points with (x, y) coordinates), with special formulas for adding and subtracting them (ie. for calculating the coordinates of $R = P + Q$), and you can also multiply a point by an integer (ie. $P \cdot n = P + P + ... + P$), though there's a much faster way to compute it if n is big).
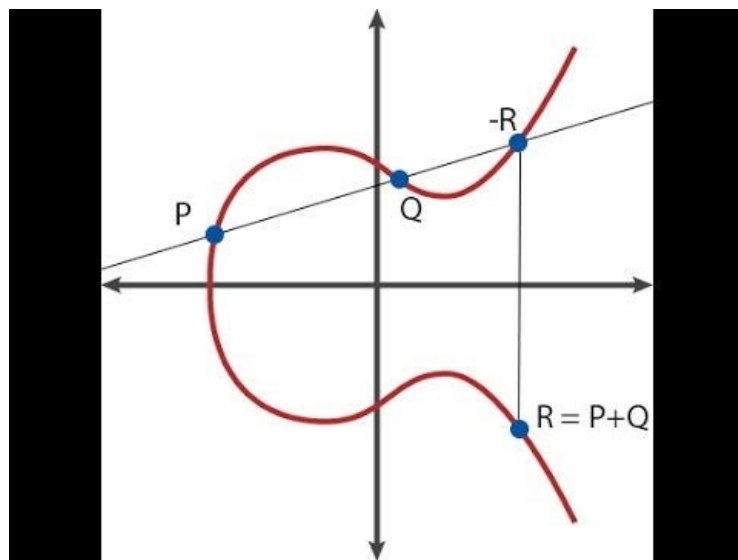
Here's how point addition looks like graphically.

There exists a special point called the "point at infinity" (O), the equivalent of zero in point arithmetic; it's always the case that $P + O = P$. Also, a curve has an "**order**"; there exists a number n such that $P \cdot n = O$ for any P (and of course, $P \cdot (n+1) = P, P \cdot (7 \cdot n + 5) = P \cdot 5$, and so on). There is also some commonly agreed upon "generator point" G, which is understood to in some sense represent the number 1. Theoretically, any point on a curve (except O) can be G; all that matters is that G is standardized.

Pairings go a step further in that they allow you to check certain kinds of more complicated equations on elliptic curve points — for example, if $P = G \cdot p, Q = G \cdot q$ and $R = G \cdot r$, you can check whether or not $p \cdot q = r$, having just P, Q and R as inputs. This might seem like the fundamental security guarantees of elliptic curves are being broken, as information about p is leaking from just knowing P, but it turns out that the leakage is highly contained — specifically, the [decisional Diffie Hellman problem](#) is easy, but the computational Diffie Hellman problem (knowing P and Q in the above example, *computing* $R = G \cdot p \cdot q$) and the [discrete logarithm problem](#) (recovering p from P) remain computationally infeasible (at least, if they were before).

A third way to look at what pairings do, and one that is perhaps most illuminating for most of the use cases that we are about, is that if you view elliptic curve points as one-way encrypted numbers (that is, $encrypt(p) = p \cdot G = P$), then whereas traditional elliptic curve math lets you check *linear* constraints on the numbers (eg. if $P = G \cdot p, Q = G \cdot q$ and $R = G \cdot r$, checking $5 \cdot P + 7 \cdot Q = 11 \cdot R$ is *really* checking that $5 \cdot p + 7 \cdot q = 11 \cdot r$), pairings let you check *quadratic* constraints (eg. checking $e(P, Q) \cdot e(G, G \cdot 5) = 1$ is *really* checking that $p \cdot q + 5 = 0$). And going up to quadratic is enough to let us work with deterministic threshold signatures, quadratic arithmetic programs and all that other good stuff.

Now, what is this funny $e(P, Q)$ operator that we introduced above? This is the pairing. Mathematicians also sometimes call it a *bilinear map*; the word "bilinear" here basically means that it satisfies the constraints:

$e(P, Q + R) = e(P, Q) \cdot e(P, R)$

$e(P + S, Q) = e(P, Q) \cdot e(S, Q)$

Note that $+$ and $\cdot$ can be arbitrary operators; when you're creating fancy new kinds of mathematical objects, abstract algebra doesn't care how $+$ and $\cdot$ are *defined*, as long as they are consistent in the usual ways, eg. $a + b = b + a, (a \cdot b) \cdot c = a \cdot (b \cdot c)$ and $(a \cdot c) + (b \cdot c) = (a + b) \cdot c$.

If P, Q, R and S were simple *numbers*, then making a simple pairing is easy: we can do $e(x, y) = 2^{xy}$. Then, we can see:

$e(3, 4+ 5) = 2^{3 \cdot 9} = 2^{27}$

$e(3, 4) \cdot e(3, 5) = 2^{3 \cdot 4} \cdot 2^{3 \cdot 5} = 2^{12} \cdot 2^{15} = 2^{27}$

It's bilinear!

However, such simple pairings are not suitable for cryptography because the objects that they work on are simple integers and are too easy to analyze; integers make it easy to divide, compute logarithms, and make various other computations; simple integers have no concept of a "public key" or a "one-way function". Additionally, with the pairing described above you can go backwards - knowing x, and knowing $e(x, y)$, you can simply compute a division and a logarithm to determine y. We want mathematical objects that are as close as possible to "black boxes", where you can add, subtract, multiply and divide, *but do nothing else*. This is where elliptic curves and elliptic curve pairings come in.

It turns out that it is possible to make a bilinear map over elliptic curve points — that is, come up with a function $e(P, Q)$ where the inputs P and Q are elliptic curve points, and where the output is what's called an $(F_p)^{12}$ element (at least in the specific case we will cover here; the specifics differ depending on the details of the curve, more on this later), but the math behind doing so is quite complex.

First, let's cover prime fields and extension fields. The pretty elliptic curve in the picture earlier in this post only looks that way if you assume that the curve equation is defined using regular real numbers. However, if we actually use regular real numbers in cryptography, then you can use logarithms to "go backwards", and everything breaks; additionally, the amount of space needed to actually store and represent the numbers may grow arbitrarily. Hence, we instead use numbers in a **prime field**.

A prime field consists of the set of numbers $0, 1, 2... p-1$, where p is prime, and the various operations are defined as follows:

$a + b: (a + b)$ % p

$a \cdot b: (a \cdot b)$ % p

$(a - b: (a - b))$ % (p)

$(a / b: (a \cdot b^{p-2}))$ % (p)

Basically, all math is done modulo (p) (see here for an introduction to modular math). Division is a special case; normally, $(\frac{3}{2})$ is not an integer, and here we want to deal only with integers, so we instead try to find the number (x) such that $(x \cdot 2 = 3)$, where $(\cdot)$ of course refers to modular multiplication as defined above. Thanks to Fermat's little theorem, the exponentiation trick shown above does the job, but there is also a faster way to do it, using the Extended Euclidean Algorithm. Suppose $(p = 7)$; here are a few examples:

$(2 + 3 = 5)$ % $(7 = 5)$

$(4 + 6 = 10)$ % $(7 = 3)$

$(2 - 5 = -3)$ % $(7 = 4)$

$(6 \cdot 3 = 18)$ % $(7 = 4)$

$(3 / 2 = (3 \cdot 2^5))$ % $(7 = 5)$

$(5 \cdot 2 = 10)$ % $(7 = 3)$

If you play around with this kind of math, you'll notice that it's perfectly consistent and satisfies all of the usual rules. The last two examples above show how $((a / b) \cdot b = a)$; you can also see that $((a + b) + c = a + (b + c), (a + b) \cdot c = a \cdot c + b \cdot c)$, and all the other high school algebraic identities you know and love continue to hold true as well. In elliptic curves in reality, the points and equations are usually computed in prime fields.

Now, let's talk about **extension fields**. You have probably already seen an extension field before; the most common example that you encounter in math textbooks is the field of complex numbers, where the field of real numbers is "extended" with the additional element $(\sqrt{-1} = i)$. Basically, extension fields work by taking an existing field, then "inventing" a new element and defining the relationship between that element and existing elements (in this case, $(i^2 + 1 = 0)$), making sure that this equation does not hold true for any number that is in the original field, and looking at the set of all linear combinations of elements of the original field and the new element that you have just created.
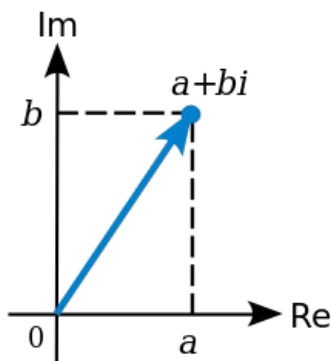
We can do extensions of prime fields too; for example, we can extend the prime field $(\bmod 7)$ that we described above with (i), and then we can do:

$((2 + 3i) + (4 + 2i) = 6 + 5i)$

$((5 + 2i) + 3 = 1 + 2i)$

$((6 + 2i) \cdot 2 = 5 + 4i)$

$(4i \cdot (2 + i) = 3 + i)$

That last result may be a bit hard to figure out; what happened there was that we first decompose the product into $(4i \cdot 2 + 4i \cdot i)$, which gives $(8i - 4)$, and then because we are working in $(\bmod 7)$ math that becomes $(i + 3)$. To divide, we do:

\(a / b: (a \cdot b^{(p^2-2)})\) % \(p\)

Note that the exponent for Fermat's little theorem is now \(p^2\) instead of \(p\), though once again if we want to be more efficient we can also instead extend the Extended Euclidean Algorithm to do the job. Note that \(x^{p^2 - 1} = 1\) for any \(x\) in this field, so we call \(p^2 - 1\) the "order of the multiplicative group in the field".

With real numbers, the [Fundamental Theorem of Algebra](#) ensures that the quadratic extension that we call the complex numbers is "complete" — you cannot extend it further, because for any mathematical relationship (at least, any mathematical relationship defined by an algebraic formula) that you can come up with between some new element \(j\) and the existing complex numbers, it's possible to come up with at least one complex number that already satisfies that relationship. With prime fields, however, we do not have this issue, and so we can go further and make cubic extensions (where the mathematical relationship between some new element \(w\) and existing field elements is a cubic equation, so \(1, w\) and \(w^2\) are all linearly independent of each other), higher-order extensions, extensions of extensions, etc. And it is these kinds of supercharged modular complex numbers that elliptic curve pairings are built on.

For those interested in seeing the exact math involved in making all of these operations written out in code, prime fields and field extensions are implemented here: https://github.com/ethereum/py_pairing/blob/master/py_ecc/bn128/bn128_field_elements.py

Now, on to elliptic curve pairings. An elliptic curve pairing (or rather, the specific form of pairing we'll explore here; there are also other types of pairings, though their logic is fairly similar) is a map \(G_2 \times G_1 \rightarrow G_t\), where:

- \(\bf G_1\) is an elliptic curve, where points satisfy an equation of the form \(y^2 = x^3 + b\), and where both coordinates are elements of \(F_p\) (ie. they are simple numbers, except arithmetic is all done modulo some prime number)

- \(\bf G_2\) is an elliptic curve, where points satisfy the same equation as \(G_1\), except where the coordinates are elements of \((F_p)^{12}\) (ie. they are the supercharged complex numbers we talked about above; we define a new "magic number" \(w\), which is defined by a \(12\)th degree polynomial like \(w^{12} - 18 \cdot w^6 + 82 = 0\))

- \(\bf G_t\) is the type of object that the result of the elliptic curve goes into. In the curves that we look at, \(G_t\) is \(\bf (F_p)^{12}\) (the same supercharged complex number as used in \(G_2\))

The main property that it must satisfy is bilinearity, which in this context means that:

- \(e(P, Q + R) = e(P, Q) \cdot e(P, R)\)

- \(e(P + Q, R) = e(P, R) \cdot e(Q, R)\)

There are two other important criteria:

- **Efficient computability** (eg. we can make an easy pairing by simply taking the discrete logarithms of all points and multiplying them together, but this is as computationally hard as breaking elliptic curve cryptography in the first place, so it doesn't count)

- **Non-degeneracy** (sure, you could just define \(e(P, Q) = 1\), but that's not a particularly useful pairing)

So how do we do this?

The math behind why pairing functions work is quite tricky and involves quite a bit of advanced algebra going even beyond what we've seen so far, but I'll provide an outline. First of all, we need to define the concept of a **divisor**, basically an alternative way of representing functions on elliptic curve points. A divisor of a function basically counts the zeroes and the infinities of the function. To see what this means, let's go through a few examples. Let us fix some point \(P = (P_x, P_y)\), and consider the following function:

\(f(x, y) = x - P_x\)

The divisor is \([P] + [-P] - 2 \cdot [O]\) (the square brackets are used to represent the fact that we are referring to *the presence of the point \(P\) in the set of zeroes and infinities of the function*, not the point P itself; \([P] + [Q]\) is **not** the same thing as \([P + Q]\)). The reasoning is as follows:

- The function is equal to zero at \(P\), since \(x\) *is* \(P_x\), so \(x - P_x = 0\)

- The function is equal to zero at $-P$, since $-P$ and P share the same x coordinate

- The function goes to infinity as x goes to infinity, so we say the function is equal to infinity at O. There's a technical reason why this infinity needs to be counted twice, so O gets added with a "multiplicity" of -2 (negative because it's an infinity and not a zero, two because of this double counting).

The technical reason is roughly this: because the equation of the curve is $x^3 = y^2 + b$, y goes to infinity "1.5 times faster" than x does in order for y^2 to keep up with x^3; hence, if a linear function includes only x then it is represented as an infinity of multiplicity 2, but if it includes y then it is represented as an infinity of multiplicity 3.

Now, consider a "line function":

$ax + by + c = 0$

Where a, b and c are carefully chosen so that the line passes through points P and Q. Because of how elliptic curve addition works (see the diagram at the top), this also means that it passes through $-P-Q$. And it goes up to infinity dependent on both x and y, so the divisor becomes $[P]+ [Q] + [-P-Q] - 3 \cdot [O]$.
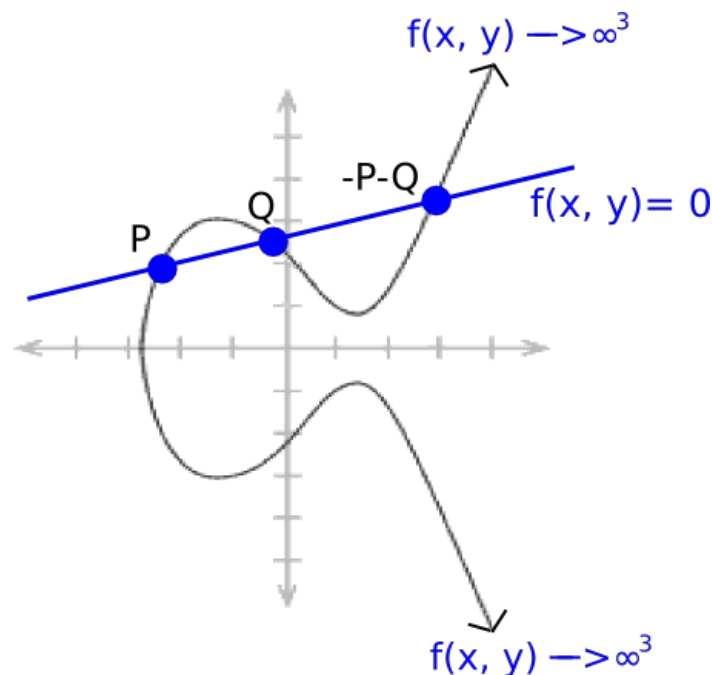
We know that every "rational function" (ie. a function defined only using a finite number of $+, -, \cdot$) and $(/)$ operations on the coordinates of the point) uniquely corresponds to some divisor, up to multiplication by a constant (ie. if two functions F and G have the same divisor, then $F = G \cdot k$ for some constant k).

For any two functions F and G, the divisor of $F \cdot G$ is equal to the divisor of F plus the divisor of G (in math textbooks, you'll see $(F \cdot G) = (F) + (G)$), so for example if $f(x, y) = P_x - x$, then $(f^3) = 3 \cdot [P] + 3 \cdot [-P] - 6 \cdot [O]$; P and $-P$ are "triple-counted" to account for the fact that f^3 approaches 0 at those points "three times as quickly" in a certain mathematical sense.

Note that there is a theorem that states that if you "remove the square brackets" from a divisor of a function, the points must add up to O ($[P] + [Q] + [-P-Q] - 3 \cdot [O]$ clearly fits, as $P + Q - P - Q - 3 \cdot O = O$), and any divisor that has this property is the divisor of a function.

Now, we're ready to look at Tate pairings. Consider the following functions, defined via their divisors:

- $(F_P) = n \cdot [P] - n \cdot [O]$, where n is the order of G_1, ie. $n \cdot P = O$ for any P

- $(F_Q) = n \cdot [Q] - n \cdot [O]$

- $(g) = [P + Q] - [P] - [Q] + [O]$

Now, let's look at the product $F_P \cdot F_Q \cdot g^n$. The divisor is:

$n \cdot [P] - n \cdot [O] + n \cdot [Q] - n \cdot [O] + n \cdot [P + Q] - n \cdot [P] - n \cdot [Q] + n \cdot [O]$

Which simplifies neatly to:

$n \cdot [P + Q] - n \cdot [O]$

Notice that this divisor is of exactly the same format as the divisor for F_P and F_Q above. Hence, $F_P \cdot F_Q \cdot g^n = F_{P + Q}$.

Now, we introduce a procedure called the "final exponentiation" step, where we take the result of our functions above (F_P, F_Q, etc.) and raise it to the power $z = (p^{12} - 1) / n$, where $p^{12} - 1$ is the order of the multiplicative group in $(F_p)^{12}$ (ie. for *any* $x \in (F_p)^{12}$, $x^{(p^{12} - 1)} = 1$). Notice that if you apply this exponentiation to any result that has *already* been raised to the power of n, you get an exponentiation to the power of $p^{12} - 1$, so the result turns into 1. Hence, after final exponentiation, g^n cancels out and we get $F_P^z \cdot F_Q^z = (F_{P + Q})^z$. There's some bilinearity for you.

Now, if you want to make a function that's bilinear in both arguments, you need to go into spookier math, where instead of taking F_P of a value directly, you take F_P of a *divisor*, and that's where the full "Tate pairing" comes from. To prove some more results you have to deal with notions like "linear equivalence" and "Weil reciprocity", and the rabbit hole goes on from there. You can find more reading material on all of this [here](#) and [here](#).

For an implementation of a modified version of the Tate pairing, called the optimal Ate paring, [see here](#). The code implements [Miller's algorithm](#), which is needed to actually compute F_P.

Note that the fact pairings like this are possible is somewhat of a mixed blessing: on the one hand, it means that all the protocols we can do with pairings become possible, but is also means that we have to be more careful about what elliptic curves we use.

Every elliptic curve has a value called an *embedding degree*; essentially, the smallest k such that $(p^k - 1)$ is a multiple of n (where p is the prime used for the field and n is the curve order). In the fields above, $k = 12$, and in the fields used for traditional ECC (ie. where we don't care about pairings), the embedding degree is often extremely large, to the point that pairings are computationally infeasible to compute; however, if we are not careful then we can generate fields where $k = 4$ or even 1.

If $k = 1$, then the "discrete logarithm" problem for elliptic curves (essentially, recovering p knowing only the point $P = G \cdot p$, the problem that you have to solve to "crack" an elliptic curve private key) can be reduced into a similar math problem over F_p, where the problem becomes much easier (this is called the [MOV attack](#)); using curves with an embedding degree of 12 or higher ensures that this reduction is either unavailable, or that solving the discrete log problem over pairing results is at least as hard as recovering a private key from a public key "the normal way" (ie. computationally infeasible). Do not worry; all standard curve parameters have been thoroughly checked for this issue.

Stay tuned for a mathematical explanation of how zk-SNARKs work, coming soon.

Special thanks to Christian Reitwiessner, Ariel Gabizon (from Zcash) and Alfred Menezes for reviewing and making corrections.

[Mirror] A Proof of Stake Design Philosophy

This is a mirror of the post at https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51

Systems like Ethereum (and Bitcoin, and NXT, and Bitshares, etc) are a fundamentally new class of cryptoeconomic organisms — decentralized, jurisdictionless entities that exist entirely in cyberspace, maintained by a combination of cryptography, economics and social consensus. They are kind of like BitTorrent, but they are also not like BitTorrent, as BitTorrent has no concept of state — a distinction that turns out to be crucially important. They are sometimes described as [decentralized autonomous corporations](#), but they are also not quite corporations — you can't hard fork Microsoft. They are kind of like open source software projects, but they are not quite that either — you can fork a blockchain, but not quite as easily as you can fork OpenOffice.

These cryptoeconomic networks come in many flavors — ASIC-based PoW, GPU-based PoW, naive PoS, delegated PoS, hopefully soon Casper PoS — and each of these flavors inevitably comes with its own underlying philosophy. One well-known example is the maximalist vision of proof of work, where "the" correct blockchain, singular, is defined as the chain that miners have burned the largest amount of economic capital to create. Originally a mere in-protocol fork choice rule, this mechanism has in many cases been elevated to a sacred tenet — see [this Twitter discussion between myself and Chris DeRose](#) for an example of someone seriously trying to defend the idea in a pure form, even in the face of hash-algorithm-changing protocol hard forks. Bitshares' [delegated proof of stake](#) presents another coherent philosophy, where everything once again flows from a single tenet, but one that can be described even more simply: [shareholders vote](#).

Each of these philosophies; Nakamoto consensus, social consensus, shareholder voting consensus, leads to its own set of conclusions and leads to a system of values that makes quite a bit of sense when viewed on its own terms — though they can certainly be criticized when compared against each other. Casper consensus has a philosophical underpinning too, though one that has so far not been as succinctly articulated.

Myself, Vlad, Dominic, Jae and others all have their own views on why proof of stake protocols exist and how to design them, but here I intend to explain where I personally am coming from.

I'll proceed to listing observations and then conclusions directly.

- Cryptography is truly special in the 21st century because **cryptography is one of the very few fields where adversarial conflict continues to heavily favor the defender**. Castles are far easier to destroy than build, islands are defendable but can still be attacked, but an average person's ECC keys are secure enough to resist even state-level actors. Cypherpunk philosophy is fundamentally about leveraging this precious asymmetry to create a world that better preserves the autonomy of the individual, and cryptoeconomics is to some extent an extension of that, except this time protecting the safety and liveness of complex systems of coordination and collaboration, rather than simply the integrity and confidentiality of private messages. **Systems that consider themselves ideological heirs to the cypherpunk spirit should maintain this basic property, and be much more expensive to destroy or disrupt than they are to use and maintain.**

- The "cypherpunk spirit" isn't just about idealism; making systems that are easier to defend than they are to attack is also simply sound engineering.

- **On medium to long time scales, humans are quite good at consensus**. Even if an adversary had access to unlimited hashing power, and came out with a 51% attack of any major blockchain that reverted even the last month of history, convincing the community that this chain is legitimate is much harder than just outrunning the main chain's hashpower. They would need to subvert block explorers, every trusted member in the community, the New York Times, archive.org, and many other sources on the internet; all in all, convincing the world that the new attack chain is the one that came first in the information technology-dense 21st century is about as hard as convincing the world that the US moon landings never happened. **These social**

considerations are what ultimately protect any blockchain in the long term, regardless of whether or not the blockchain's community admits it ([note that](#) Bitcoin Core [does admit](#) this primacy of the social layer).

- However, a blockchain protected by social consensus alone would be far too inefficient and slow, and too easy for disagreements to continue without end (though despite all difficulties, [it has happened](#)); hence, **economic consensus serves an extremely important role in protecting liveness and safety properties in the short term.**

- Because proof of work security can only come from block rewards (in Dominic Williams' terms, it [lacks two of the three Es](#)), and incentives to miners can only come from the risk of them losing their future block rewards, **proof of work necessarily operates on a logic of massive power incentivized into existence by massive rewards**. Recovery from attacks in PoW is very hard: the first time it happens, you can hard fork to change the PoW and thereby render the attacker's ASICs useless, but the second time you no longer have that option, and so the attacker can attack again and again. Hence, the size of the mining network has to be so large that attacks are inconceivable. Attackers of size less than X are discouraged from appearing by having the network constantly spend X every single day. **I reject this logic because (i) it [kills trees](#), and (ii) it fails to realize the cypherpunk spirit — cost of attack and cost of defense are at a 1:1 ratio, so there is no defender's advantage**.

- **Proof of stake breaks this symmetry by relying not on rewards for security, but rather penalties**. Validators put money ("deposits") at stake, are rewarded slightly to compensate them for locking up their capital and maintaining nodes and taking extra precaution to ensure their private key safety, but the bulk of the cost of reverting transactions comes from penalties that are hundreds or thousands of times larger than the rewards that they got in the meantime. **The "one-sentence philosophy" of proof of stake is thus not "security comes from burning energy", but rather "security comes from putting up economic value-at-loss"**. A given block or state has $X security if you can prove that achieving an equal level of finalization for any conflicting block or state cannot be accomplished unless malicious nodes complicit in an attempt to make the switch pay $X worth of in-protocol penalties.

- Theoretically, a majority collusion of validators may take over a proof of stake chain, and start acting maliciously. However, (i) through clever protocol design, their ability to earn extra profits through such manipulation can be limited as much as possible, and more importantly (ii) if they try to prevent new validators from joining, or execute 51% attacks, then the community can simply coordinate a hard fork and delete the offending validators' deposits. **A successful attack may cost $50 million, but the process of cleaning up the consequences will not be *that* much more onerous than the [geth/parity consensus failure of 2016.11.25](#)**. Two days later, the blockchain and community are back on track, attackers are $50 million poorer, and the rest of the community is likely richer since the attack will have caused the value of the token to go *up* due to the ensuing supply crunch. *That's* attack/defense asymmetry for you.

- The above should not be taken to mean that unscheduled hard forks will become a regular occurrence; if desired, the cost of a *single* 51% attack on proof of stake can certainly be set to be as high as the cost of a *permanent* 51% attack on proof of work, and the sheer cost and ineffectiveness of an attack should ensure that it is almost never attempted in practice.

- **Economics is not everything**. Individual actors may be motivated by extra-protocol motives, they may get hacked, they may get kidnapped, or they may simply get drunk and decide to wreck the blockchain one day and to hell with the cost. Furthermore, on the bright side, **individuals' moral forbearances and communication inefficiencies will often raise the cost of an attack to levels much higher than the nominal protocol-defined value-at-loss**. This is an advantage that we cannot rely on, but at the same time it is an advantage that we should not needlessly throw away.

- **Hence, the best protocols are protocols that work well under a variety of models and assumptions** — economic rationality with coordinated choice, economic rationality with individual choice, simple fault tolerance, Byzantine fault tolerance (ideally both the adaptive and non-adaptive adversary variants), [Ariely/Kahneman-inspired behavioral economic models](#) ("we all cheat just a little") and ideally any other model that's realistic and practical to reason about. **It is important to have both layers of defense: economic incentives to discourage centralized cartels from acting anti-socially, and anti-centralization incentives to discourage cartels from forming in the first place.**

- **Consensus protocols that work as-fast-as-possible have risks and should be approached**

very carefully if at all, because if the *possibility* to be very fast is tied to *incentives* to do so, the combination will reward very high and systemic-risk-inducing levels of **network-level centralization** (eg. all validators running from the same hosting provider). Consensus protocols that don't care too much how fast a validator sends a message, as long as they do so within some acceptably long time interval (eg. 4–8 seconds, as we empirically know that latency in ethereum is usually ~500ms-1s) do not have these concerns. A possible middle ground is creating protocols that can work very quickly, but where mechanics similar to Ethereum's uncle mechanism ensure that the marginal reward for a node increasing its degree of network connectivity beyond some easily attainable point is fairly low.

From here, there are of course many details and many ways to diverge on the details, but the above are the core principles that at least my version of Casper is based on. From here, we can certainly debate tradeoffs between competing values . Do we give ETH a 1% annual issuance rate and get an $50 million cost of forcing a remedial hard fork, or a zero annual issuance rate and get a $5 million cost of forcing a remedial hard fork? When do we increase a protocol's security under the economic model in exchange for decreasing its security under a fault tolerance model? Do we care more about having a predictable level of security or a predictable level of issuance? These are all questions for another post, and the various ways of *implementing* the different tradeoffs between these values are questions for yet more posts. But we'll get to it :)

# [Mirror] Quadratic Arithmetic Programs: from Zero to Hero

2016 Dec 10

This is a mirror of the post at https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649

There has been a lot of interest lately in the technology behind zk-SNARKs, and people are increasingly [trying to demystify](#) something that many have come to call "moon math" due to its perceived sheer indecipherable complexity. zk-SNARKs are indeed quite challenging to grasp, especially due to the sheer number of moving parts that need to come together for the whole thing to work, but if we break the technology down piece by piece then comprehending it becomes simpler.

The purpose of this post is not to serve as a full introduction to zk-SNARKs; it assumes as background knowledge that (i) you know what zk-SNARKs are and what they do, and (ii) know enough math to be able to reason about things like polynomials (if the statement $P(x) + Q(x) = (P + Q)(x)$, where P and Q are polynomials, seems natural and obvious to you, then you're at the right level). Rather, the post digs deeper into the machinery behind the technology, and tries to explain as well as possible the first half of the pipeline, as drawn by zk-SNARK researcher Eran Tromer here:
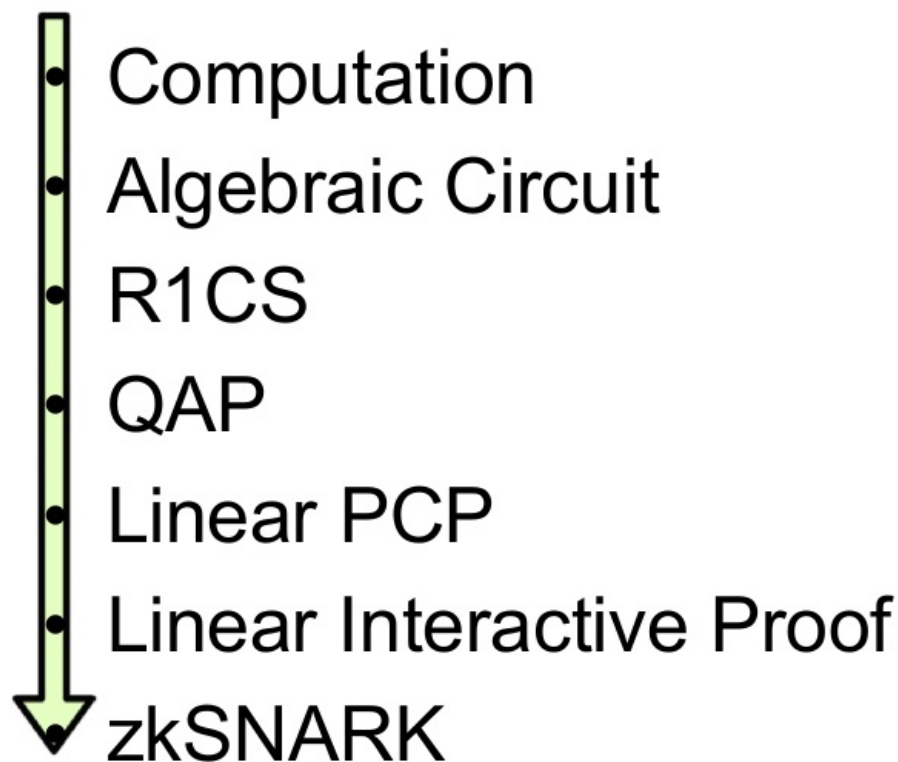
The steps here can be broken up into two halves. First, zk-SNARKs cannot be applied to any computational problem directly; rather, you have to convert the problem into the right "form" for the problem to operate on. The form is called a "quadratic arithmetic program" (QAP), and transforming the code of a function into one of these is itself highly nontrivial. Along with the process for converting the code of a function into a QAP is another process that can be run alongside so that if you have an input to the code you can create a corresponding solution (sometimes called "witness" to the QAP). After this, there is another fairly intricate process for creating the actual "zero knowledge proof" for this witness, and a separate process for verifying a proof that someone else passes along to you, but these are details that are out of scope for this post.

The example that we will choose is a simple one: proving that you know the solution to a cubic

equation: $x^3 + x + 5 = 35$ (hint: the answer is 3). This problem is simple enough that the resulting QAP will not be so large as to be intimidating, but nontrivial enough that you can see all of the machinery come into play.

Let us write out our function as follows:

```
def qeval(x):
    y = x**3
    return x + y + 5
```

The simple special-purpose programming language that we are using here supports basic arithmetic ($+$, $-$, $\cdot$, $/$), constant-power exponentiation (x^7 but not x^y) and variable assignment, which is powerful enough that you can theoretically do any computation inside of it (as long as the number of computational steps is bounded; no loops allowed). Note that modulo (%) and comparison operators ($<$, $>$, $\leq$, $\geq$) are NOT supported, as there is no efficient way to do modulo or comparison directly in finite cyclic group arithmetic (be thankful for this; if there was a way to do either one, then elliptic curve cryptography would be broken faster than you can say "binary search" and "Chinese remainder theorem").

You can extend the language to modulo and comparisons by providing bit decompositions (eg. $13 = 2^3 + 2^2 + 1$) as auxiliary inputs, proving correctness of those decompositions and doing the math in binary circuits; in finite field arithmetic, doing equality (==) checks is also doable and in fact a bit easier, but these are both details we won't get into right now. We can extend the language to support conditionals (eg. if $x < 5: y = 7;$ else: $y = 9$) by converting them to an arithmetic form: $y = 7 \cdot (x < 5) + 9 \cdot (x \geq 5)$ though note that both "paths" of the conditional would need to be executed, and if you have many nested conditionals then this can lead to a large amount of overhead.

Let us now go through this process step by step. If you want to do this yourself for any piece of code, I [implemented a compiler here](#) (for educational purposes only; not ready for making QAPs for real-world zk-SNARKs quite yet!).

Flattening

The first step is a "flattening" procedure, where we convert the original code, which may contain arbitrarily complex statements and expressions, into a sequence of statements that are of two forms: $x = y$ (where y can be a variable or a number) and $x = y$ (op) z (where op can be $+$, $-$, $\cdot$, $/$ and y and z can be variables, numbers or themselves sub-expressions). You can think of each of these statements as being kind of like logic gates in a circuit. The result of the flattening process for the above code is as follows:

```
sym_1 = x * x
y = sym_1 * x
sym_2 = y + x
~out = sym_2 + 5
```

If you read the original code and the code here, you can fairly easily see that the two are equivalent.

Gates to R1CS

Now, we convert this into something called a rank-1 constraint system (R1CS). An R1CS is a sequence of groups of three vectors (a, b, c), and the solution to an R1CS is a vector s, where s must satisfy the equation $s . a \cdot s . b - s . c = 0$, where $.$ represents the dot product - in simpler terms, if we "zip together" a and s, multiplying the two values in the same positions, and then take the sum of these products, then do the same to b and s and then c and s, then the third result equals the product of the first two results. For example, this is a satisfied R1CS:

$$A \quad B \quad C$$

A		B		C	
1	5	1	1	1	0
3	0	3	0	3	0
35	0	35	0	35	1
9	0	9	0	9	0
27	0	27	0	27	0
30	1	30	0	30	0

$$35 \quad * \quad 1 \quad - \quad 35 \quad = \quad 0$$

But instead of having just one constraint, we are going to have many constraints: one for each logic gate. There is a standard way of converting a logic gate into a (a, b, c) triple depending on what the operation is ($+$, $-$, $\cdot$ or $/$) and whether the arguments are variables or numbers. The length of each vector is equal to the total number of variables in the system, including a dummy variable ~one at the first index representing the number 1, the input variables, a dummy variable ~out representing the output, and then all of the intermediate variables (sym_1 and sym_2 above); the vectors are generally going to be very sparse, only filling in the slots corresponding to the variables that are affected by some particular logic gate.

First, we'll provide the variable mapping that we'll use:

```
'~one', 'x', '~out', 'sym_1', 'y', 'sym_2'
```

The solution vector will consist of assignments for all of these variables, in that order.

Now, we'll give the (a, b, c) triple for the first gate:

```
a = [0, 1, 0, 0, 0, 0]
b = [0, 1, 0, 0, 0, 0]
c = [0, 0, 0, 1, 0, 0]
```

You can see that if the solution vector contains 3 in the second position, and 9 in the fourth position, then regardless of the rest of the contents of the solution vector, the dot product check will boil down to $3 \cdot 3 = 9$, and so it will pass. If the solution vector has -3 in the second position and 9 in the fourth position, the check will also pass; in fact, if the solution vector has 7 in the second position and 49 in the fourth position then that check will still pass — the purpose of this first check is to verify the consistency of the inputs and outputs of the first gate only.

Now, let's go on to the second gate:

```
a = [0, 0, 0, 1, 0, 0]
b = [0, 1, 0, 0, 0, 0]
c = [0, 0, 0, 0, 1, 0]
```

In a similar style to the first dot product check, here we're checking that $sym\_1 \cdot x = y$.

Now, the third gate:

```
a = [0, 1, 0, 0, 1, 0]
b = [1, 0, 0, 0, 0, 0]
c = [0, 0, 0, 0, 0, 1]
```

Here, the pattern is somewhat different: it's multiplying the first element in the solution vector by the second element, then by the fifth element, adding the two results, and checking if the sum equals the sixth element. Because the first element in the solution vector is always one, this is just an addition check, checking that the output equals the sum of the two inputs.

Finally, the fourth gate:

```
a = [5, 0, 0, 0, 0, 1]
b = [1, 0, 0, 0, 0, 0]
c = [0, 0, 1, 0, 0, 0]
```

Here, we're evaluating the last check, ~out $(= sym\_2 + 5)$. The dot product check works by taking the sixth element in the solution vector, adding five times the first element (reminder: the first element is (1), so this effectively means adding (5)), and checking it against the third element, which is where we store the output variable.

And there we have our R1CS with four constraints. The witness is simply the assignment to all the variables, including input, output and internal variables:

```
[1, 3, 35, 9, 27, 30]
```

You can compute this for yourself by simply "executing" the flattened code above, starting off with the input variable assignment $(x=3)$, and putting in the values of all the intermediate variables and the output as you compute them.

The complete R1CS put together is:

```
A
[0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 1, 0]
[5, 0, 0, 0, 0, 1]

B
[0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]

C
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0]
```

R1CS to QAP

The next step is taking this R1CS and converting it into QAP form, which implements the exact same logic except using polynomials instead of dot products. We do this as follows. We go 3from four groups of three vectors of length six to six groups of three degree-3 polynomials, where evaluating the polynomials at each *x coordinate* represents one of the constraints. That is, if we evaluate the polynomials at $(x=1)$, then we get our first set of vectors, if we evaluate the polynomials at $(x=2)$, then we get our second set of vectors, and so on.

We can make this transformation using something called a *Lagrange interpolation*. The problem that a Lagrange interpolation solves is this: if you have a set of points (ie. $((x, y))$ coordinate pairs), then doing a Lagrange interpolation on those points gives you a polynomial that passes through all of those points. We do this by decomposing the problem: for each (x) coordinate, we create a polynomial that has the desired (y) coordinate at that (x) coordinate and a (y) coordinate of (0) at all the other (x) coordinates we are interested in, and then to get the final result we add all of the polynomials together.

Let's do an example. Suppose that we want a polynomial that passes through $((1, 3), (2, 2))$ and $((3, 4))$. We start off by making a polynomial that passes through $((1, 3), (2, 0))$ and $((3, 0))$. As it turns out, making a polynomial that "sticks out" at $(x=1)$ and is zero at the other points of interest is easy; we simply do:

```
(x - 2) * (x - 3)
```

Which looks like this:
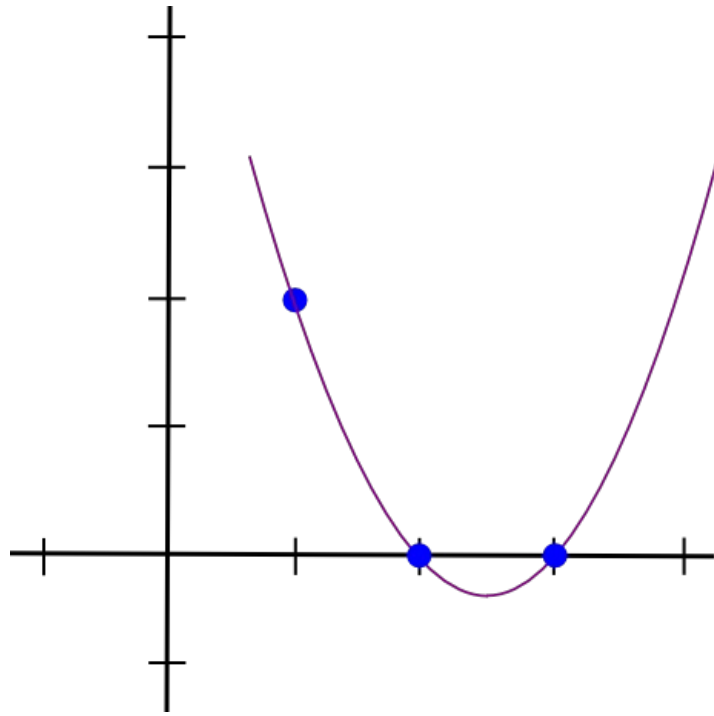
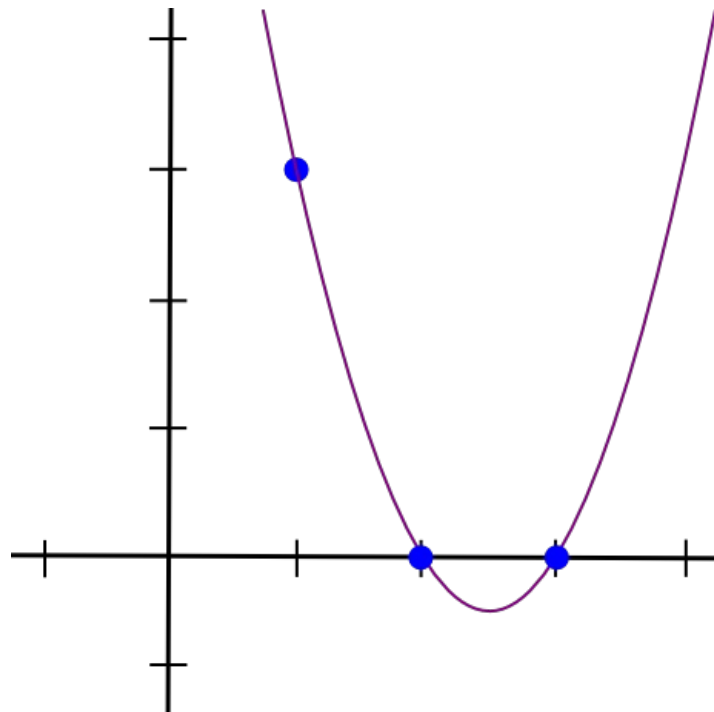Now, we just need to "rescale" it so that the height at x=1 is right:

```
(x - 2) * (x - 3) * 3 / ((1 - 2) * (1 - 3))
```
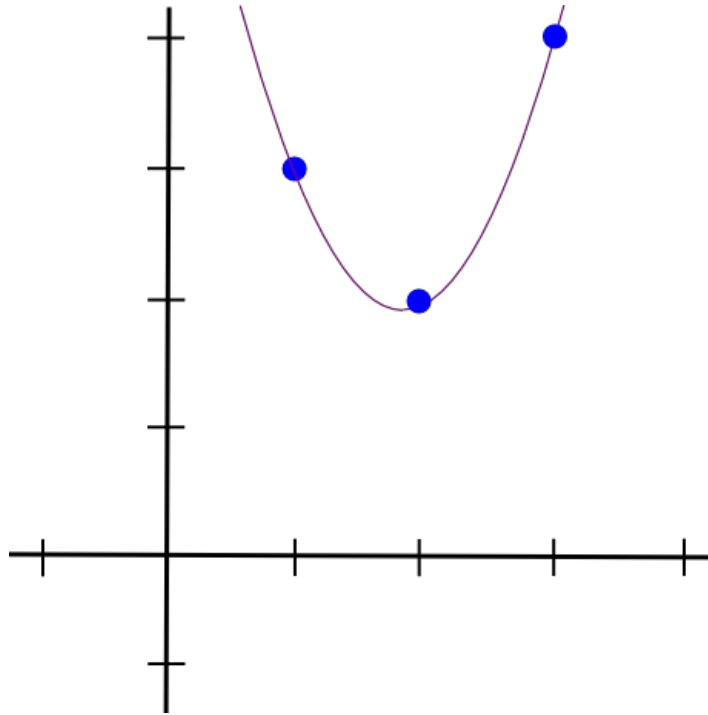
This gives us:

```
1.5 * x**2 - 7.5 * x + 9
```


We then do the same with the other two points, and get two other similar-looking polynomials, except that they "stick out" at $(x=2)$ and $(x=3)$ instead of $(x=1)$. We add all three together and get:

```
1.5 * x**2 - 5.5 * x + 7
```

With exactly the coordinates that we want. The algorithm as described above takes $O(n^3)$ time, as there are n points and each point requires $O(n^2)$ time to multiply the polynomials together; with a little thinking, this can be reduced to $O(n^2)$ time, and with a lot more thinking, using fast Fourier transform algorithms and the like, it can be reduced even further — a crucial optimization when functions that get used in zk-SNARKs in practice often have many thousands of gates.

Now, let's use Lagrange interpolation to transform our R1CS. What we are going to do is take the first value out of every a vector, use Lagrange interpolation to make a polynomial out of that (where evaluating the polynomial at i gets you the first value of the ith a vector), repeat the process for the first value of every b and c vector, and then repeat that process for the second values, the third, values, and so on. For convenience I'll provide the answers right now:

```
A polynomials
[-5.0, 9.166, -5.0, 0.833]
[8.0, -11.333, 5.0, -0.666]
[0.0, 0.0, 0.0, 0.0]
[-6.0, 9.5, -4.0, 0.5]
[4.0, -7.0, 3.5, -0.5]
[-1.0, 1.833, -1.0, 0.166]

B polynomials
[3.0, -5.166, 2.5, -0.333]
[-2.0, 5.166, -2.5, 0.333]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]

C polynomials
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[-1.0, 1.833, -1.0, 0.166]
[4.0, -4.333, 1.5, -0.166]
[-6.0, 9.5, -4.0, 0.5]
[4.0, -7.0, 3.5, -0.5]
```

Coefficients are in ascending order, so the first polynomial above is actually $0.833 \cdot x^3 - 5 \cdot x^2 + 9.166 \cdot x - 5$. This set of polynomials (plus a Z polynomial that I will explain later) makes up the parameters for this particular QAP instance. Note that all of the work up until this point needs to be done only once for every function that you are trying to use zk-SNARKs to verify; once the QAP parameters are generated, they can be reused.

Let's try evaluating all of these polynomials at $x=1$. Evaluating a polynomial at $x=1$ simply means adding up all the coefficients (as $1^k = 1$ for all k), so it's not difficult. We get:

```
A results at x=1
0
1
0
0
0
0

B results at x=1
0
1
0
0
0
0

C results at x=1
0
0
0
1
0
0
```

And lo and behold, what we have here is exactly the same as the set of three vectors for the first logic gate that we created above.

Checking the QAP

Now what's the point of this crazy transformation? The answer is that instead of checking the constraints in the R1CS individually, we can now check *all of the constraints at the same time* by doing the dot product check *on the polynomials*.

$$A(x) \ast B(x) - C(x) = H \ast Z(x)$$
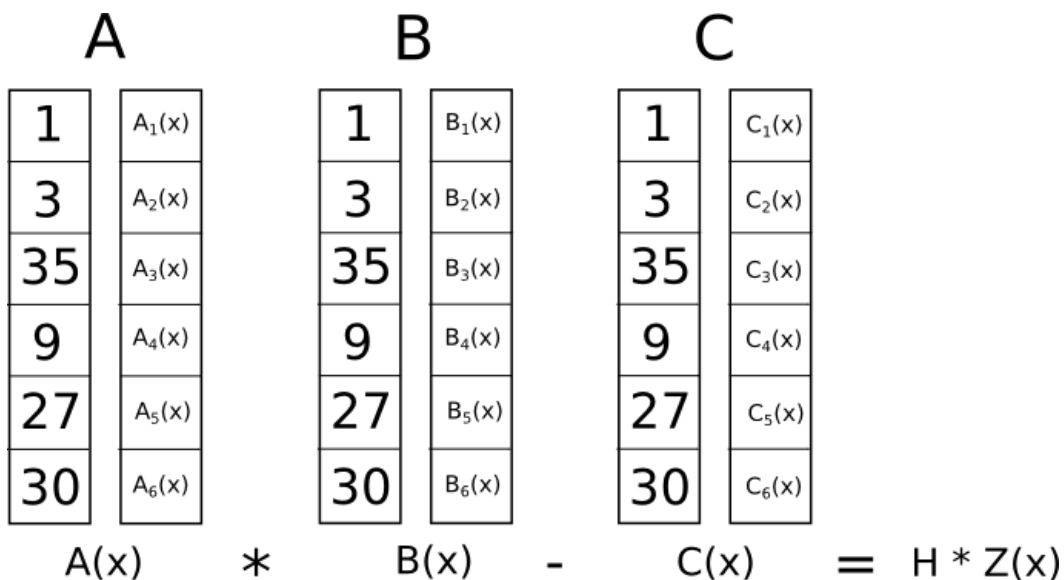
Because in this case the dot product check is a series of additions and multiplications of polynomials, the result is itself going to be a polynomial. If the resulting polynomial, evaluated at every x coordinate that we used above to represent a logic gate, is equal to zero, then that means that all of the checks pass; if the resulting polynomial evaluated at at least one of the x coordinate representing a logic gate gives a nonzero value, then that means that the values going into and out of that logic gate are inconsistent (ie. the gate is $y = x \cdot sym\_1$) but the provided values might be $x = 2, sym\_1 = 2$) and $y = 5$)).

Note that the resulting polynomial does not itself have to be zero, and in fact in most cases won't be; it could have any behavior at the points that don't correspond to any logic gates, as long as the result

is zero at all the points that *do* correspond to some gate. To check correctness, we don't actually evaluate the polynomial $t = A . s \cdot B . s - C . s$ at every point corresponding to a gate; instead, we divide t by another polynomial, Z, and check that Z evenly divides t - that is, the division t / Z leaves no remainder.

Z is defined as $(x - 1) \cdot (x - 2) \cdot (x - 3) ...$ - the simplest polynomial that is equal to zero at all points that correspond to logic gates. It is an elementary fact of algebra that *any* polynomial that is equal to zero at all of these points has to be a multiple of this minimal polynomial, and if a polynomial is a multiple of Z then its evaluation at any of those points will be zero; this equivalence makes our job much easier.

Now, let's actually do the dot product check with the polynomials above. First, the intermediate polynomials:

```
A . s = [43.0, -73.333, 38.5, -5.166]
B . s = [-3.0, 10.333, -5.0, 0.666]
C . s = [-41.0, 71.666, -24.5, 2.833]
```

Now, $A . s \cdot B . s — C . s$:

```
t = [-88.0, 592.666, -1063.777, 805.833, -294.777, 51.5, -3.444]
```

Now, the minimal polynomial $Z = (x - 1) \cdot (x - 2) \cdot (x - 3) \cdot (x - 4)$:

```
Z = [24, -50, 35, -10, 1]
```

And if we divide the result above by Z, we get:

```
h = t / Z = [-3.666, 17.055, -3.444]
```

With no remainder.

And so we have the solution for the QAP. If we try to falsify any of the variables in the R1CS solution that we are deriving this QAP solution from — say, set the last one to 31 instead of 30, then we get a t polynomial that fails one of the checks (in that particular case, the result at $x=3$ would equal -1 instead of 0), and furthermore t would not be a multiple of Z; rather, dividing t / Z would give a remainder of $[-5.0, 8.833, -4.5, 0.666]$.

Note that the above is a simplification; "in the real world", the addition, multiplication, subtraction and division will happen not with regular numbers, but rather with finite field elements — a spooky kind of arithmetic which is self-consistent, so all the algebraic laws we know and love still hold true, but where all answers are elements of some finite-sized set, usually integers within the range from 0 to $n-1$ for some n. For example, if $n = 13$, then $1 / 2 = 7$ (and $7 \cdot 2 = 1$, $3 \cdot 5 = 2$), and so forth. Using finite field arithmetic removes the need to worry about rounding errors and allows the system to work nicely with elliptic curves, which end up being necessary for the rest of the zk-SNARK machinery that makes the zk-SNARK protocol actually secure.

Special thanks to Eran Tromer for helping to explain many details about the inner workings of zk-SNARKs to me.