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

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 slowPython) 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 areactive 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 likedecentralization 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.

1. 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.

1. 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 carformally 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 tohave 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 yettype 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.