The World of ‘Trust-Minimized’ Middleware: Consensus Verification & Bridging

Fenbushi Capital | 分布式资本

Follow

--

Listen

Share

Thanks

[Brian Retford

](https://twitter.com/BrianRetford),

[Wei Dai

](https://twitter.com/_weidai),

[John Guibas

](https://twitter.com/jtguibas),

[Mo Dong

](https://twitter.com/no89thkey)and

[Mustafa Al-Bassam

](https://twitter.com/musalbas)for the great discussion that contributed to some of the ideas mentioned in this article! And thanks

[Garvit Goel

](https://twitter.com/garvitgoel03),

[Federico Carrone

](https://twitter.com/fede_intern),

[Seun Lanlege

](https://twitter.com/seunlanlege),

[Bo Du

](https://twitter.com/0xshake),

[Miko Matsumura

](https://twitter.com/mikojava),

[Misha Komarov

](https://twitter.com/nemothenoone)and

[Daniel Tehrani

](https://twitter.com/dan_tehrani), for reviewing my article!

Original article link:
https://mirror.xyz/0xE21b1e6f471EDeF18264e9BBe51b7fA7643EE6B5/3aUxlscgXnIJktkDzerlcYbVMqOzx60DdyXtiFqMaWw

Introduction

Over the past few months, some crypto chads have openly discussed the meaning behind “trust-minimization”, a term that

was slapped onto almost all of the new and flashy middleware/infra that comes with optimistic or ZKP for security. And the discussions around security assumptions behind trust-minimized bridges have spurred further discourse to reach a consensus on what that term really means.

Following a trail of relevant articles, I also found many interesting pieces, such as byChainlink and Vitalik, talking exactly about degrees of trust-assumptions. However, I noticed that there are still many significant semantic differences across the board when using the term “trust-minimization”, so I would like to share how I construe the term “trust-minimization” and why that is important in the context of blockchain middleware, with the hope that we all can finally reach consensus on what “trust-minimization” means.

Consensus on the term “trust-minimization”

Trust is an assumption about one’s behaviour holding true to the expectation. Trust-minimization in the context of blockchain refers to a general set of approaches to reducing the set of trusted components so that the outcome is more predictable. And there are different ways to achieve trust-minimization.

Vitalik has well-illustrated the levels of trust in the system by quantifying the minimum number of honest agents for the system to behave as expected. In this spectrum, the red (N-of-N) is the most trusted (where you have to trust everyone) while the bright green (0-of-N) is the most trust-minimized (where you only have to trust yourself). Most blockchain and some major middlewares, like oracles and bridges, sit in the “Middle zone” of M-of-N trust and calls themselves “trustless” given enough decentralization and crypto-economic security. This framework is a great way to contextualize trust assumptions, but it can also be misleading without context: trust-minimization is a relative term

and we have to consider the context for better assessing the trust. For example, both Ethereum and Axelar are secured with honest-majority assumptions and can be seen as “trustless” independently, but using Axelar on Ethereum would need additional trust given that the AXL token has less crypto-economic security than ETH and that Axelar requires a higher threshold of honest validators to be “secure”, hence generally considered a trusted bridge. This applies to almost all of the PoS-type of middleware built on top of Ethereum because none besides BTC has a crypto-economic mass larger than Ethereum and the cost of an attack on individual middleware will be many magnitudes lower than that of attacking Ethereum. Even the shared security model via Eigenlayer is not a perfect solution to this trust problem since participating in restaking doesn’t grant the receiving middleware the entire ETH crypto-economic mass. Strictly speaking, any PoS type of middleware on Ethereum is just another form of added M-of-N trust

, which often become the weaker security spot. Hence anything that can be considered more trust-minimized on Ethereum has to be a few-of-N

or 1-of-N

trust where it adds little to no trust overhead on existing security offered by the underlying chain.

Dissecting the approaches for trust-minimization

There are ongoing discussions and innovations that leverage both few-of-N and 1-of-N trust to further bolster the on-chain security. Here, I will discuss some of the notable examples for each different trust model.

-Few-of-N

- Data Availability Sampling (DAS) is a way to make sure that the data is available for each newly proposed block for the chain. Instead of each node hoarding all of the block data, they will be holding a small randomly selected chunk of the whole data. And with the help of erasure coding with redundant information, nodes will be able to sample a few chunks of data to be confident that the data availability of the chain. For more details, can check out here.
- Random sampling of validators’ signatures for theCasper FFG vote to estimate the entire Ethereum consensus verification. The goal here is to verify the full consensus of Ethereum without actually verifying the signatures of the entire validator set. The general random sampling problem could be categorized under the few-of-N trust as it requires only a smaller subset of honest entities to correctly estimate the behaviour of a network that relies on an honest supermajority. The Ethereum light client is also taking this approach via the sync committee, and I will discuss its underlying security assumptions later on in more detail; this type of few-of-N trust is NOT

trust-minimized.

-1-of-N

- Optimistic approach with fraud-proof/dispute allows observers to have an optimistic view of the system and assert any dispute within the dispute time delay (DTD). The optimistic model has already been widely deployed for block relaying, rollups, bridges, oracles, and more. It brings little computational overhead but may result in additional latency to resolve disputes. However, its dispute mechanisms may end up relying on the crypto-economic securities via voting or disincentivization scheme via bonding, therefore, an optimistic approach based on dispute resolution via voting could

be considered M-of-N security and is not actually trust-minimizing in some cases; an adversary could perform 51% attack on the vote based on a particular native token. Conversely, the bonding approach could be harder to collude as it would require the attacker to know and convince a highly dynamic set of validators/watchers to not submit a bonded fraud-proof. In summary, not all optimistic models are trust-minimized.

- ZK approach allows any provers to generate avalidity proof to verify the validity of a statement, such as a consensus state. This approach at its core is reliance on the correctness of the mathematical proof without any economic incentives as an intermediator. Unlike some optimistic models, the incentive is given out to provers for guaranteeing the liveness, not the security. This maximally minimizes the attack surfaces of system collusion, leaving only the system liveness as a major attack vector (system liveness is trivial as the system only needs at least one incentivized prover), truly enabling 1-of-N trust for security. One caveat on the trust-assumption is the honesty assumption during the trusted setup, which is required to generate the common reference string (CRS), a common knowledge shared between provers and verifiers to provide non-interactive proof. This is important because broken CRS will allow the prover to know the "Tau" (the secret hidden in the random) and always generate valid proofs. Though this may be less of a problem with STARK and some SNARK (Spartan, Halo2 etc.) as it relies on publicly verifiable randomness to generate the shared knowledge (public parameters). In either case, I believe ZKP is the end goal of 1-of-N trust, and there is more and more trust-minimized middleware leveraging it entering this space.

Verifying the Ethereum state and data

Most of the ZK middlewares to date are using ZKP for performing succinct verification of Ethereum state/data applied across different contexts, not actually using the Zero Knowledge aspect. Bridging/oracle (Succinct, Electron Lab, Brevis by Celer, Polymer Labs …) and co-processing/storage proof (Axiom, Relic, Hyper Oracle, Herodotus, Lagrange…) are some of the notable use cases. While each application uses ZKP in different ways, all of them are trying to do a somewhat similar thing: verifying the Ethereum state and data (some include other chains too ofc). Let's cut into the topic starting with bridging, one of the most notoriously debated middleware.

Most zkBridges (consensus proof bridges lol) behave like an output oracle using a light client to fetch and verify the block header of the corresponding chains. In this case, a succinct proof would be used to abstract away the signature verifications for two reasons.

1. Reducing the cost of signature verification: EVM does not have pre-compiles for some common signatures, such as BLS with BLS12–381 curve & EdDSA on ED25519 (Cosmos, Near & Solana). Using SNARK-based proof will allow the verification of the signatures with a relatively small footprint.

2. Superior security: Abstracting away the signature verification should come with little to no additional security overhead (security assumptions may stem from the trusted setup for SNARK). And ZKP achieves that by leaving the trust in math, not humans (OP is this).

All bridges are trying to verify and relay the most accurate consensus state to the corresponding destination domain. Hence consensus verification becomes extremely important for bridge security. And often times, such verifications are done via consensus light clients, delivering the "correct" chain state via blockheader, which begs the question of how secure the consensus light client really is. But before we dive into more details, I would like to introduce a few approaches taken by projects to build more trust-minimized bridges so that we can better contextualize the usage and the importance of light clients.

*I use consensus light client and light client synonymously for this article.

Succinct:

Succinct is building Telepathy, an interoperability protocol for Ethereum, whose security is hinged on Ethereum's light client protocol based on the sync committee (for now), a randomly selected group of 512 Ethereum validators that are reshuffled every 27 hours. Telepathy's light client contract on the destination chain will fetch and keep track of the Ethereum block header verified by >2/3rds of the sync committee validators via the Ethereum light client protocol. And instead of verifying the signatures of the committee validators natively on-chain, which is gas-inefficient, SNARK (Groth16 for now) was used to reduce the verification cost.

One issue with this approach is the on-chain storage of the public key, which is needed for the public input to the on-chain verification of the proof. Ethereum uses simple serialization (SSZ) for the 512 public keys of the validators. However, SSZ consists of many SHA256 hash functions and is not ZK-friendly due to the overhead in proving and verifying every single bitwise operation for large bit-size numbers, 256 bits in this case. This leads to inefficient on-chain verification due to large circuit sizes for SNARK as a result of the bitwise operations bloating the circuit. Therefore, commitments to public keys are made using Poseidon hash which results in a ZK-friendly representation that can be stored and used to verify SNARK efficiently on-chain. This is essentially an application of the hash-and-prove model. Few other protocols have also proposed a similar design as Succinct, such as Brevis by Celer Network

Due to the security concerns around sync-committee-based light clients, Succinct Lab has also been developing ZK light clients for consensus proof with a more optimized and scalable consensus verification scheme for Ethereum via recursive composition of proofs using Plonky2 + Starky. I will further discuss different designs that we may see across ZK light clients

later in the article.

Electron Labs:

Electron Labs originally built ZK light clients of Cosmos chains on Ethereum where verification of hundreds of ED25519 signatures will be required (the more validators on the Cosmos chain, the more expensive). As I mentioned previously, such signature verification would be prohibitively expensive without the pre-compiles. Therefore Electron Labs built out Circom ED25519 which enables batch verification of ED25519 via SNARK (Groth16). The rough outline of the design is shown in the diagram.

One of the challenges that come with this approach was that ED25519 has a bigger finite field than the alt_bn128/BN128, paring-friendly curve and finite field that are commonly used for SNARKs. Such non-native field arithmetic can be problematic because the size of the field determines the size of the arithmetic operations that are performed in the SNARKs. Arithmetic operations are performed using the R1CS (Rank-1 Constraint System), which involves reducing the problem to a set of constraints that can be solved using linear algebra over finite fields. The alt_bn128/BN128 field is a finite field of size 2^{128}, which is chosen specifically because it is large enough to perform complex arithmetic operations but small enough to ensure that the ZKP can be efficiently constructed and verified. When the finite field of the signature scheme is larger than the alt_bn128 field, it becomes more difficult to perform the necessary arithmetic operations in the ZKP, which can lead to larger and more complex circuits that are more difficult to verify. Electron Lab proposed a workaround to this problem via emulated field where they define the ED25519 finite field on base 2^{85}, which allows fast multiplication and modulo operations while keeping all the field elements within the alt-bn128/BN128 finite field. As such, SNARK proof on ED25519 can be efficiently constructed and verified.

Another issue is the proving time. It takes about 16mins to generate proof for a cosmos chain with 100 validators, which is way too long to be practical. While they can split up the batches into smaller groups and generate multiple proofs in parallel, it will result in higher gas costs from on-chain verifications.

To improve the proving time, Electron Lab suggested to:

1. Move away from SNARKs towards STARKs to enable proof generation in the browser as one way to achieve parallelization. Note that CRS (proving key) of SNARKs can be several GBs in size, and it is difficult to perform client-side proving.
2. Perform STARK-based proof recursions to aggregate proofs into a single proof
3. Use hardware acceleration.
4. Reduce the on-chain footprint via SNARK

Currently, Electron lab is also working on an Ethereum zk light client based on Plonky2 + Starky and claims to support a full consensus verification at a cost of ~$0.05 on L2. Will discuss more about the light client designs later.

Polymer Labs:

Polymer Labs are building the IBC transport hub for all major chains across the space. In traditional IBC, light clients will sit across all integrated chains running Terdemint consensus where the clients can track the consensus state of the other blockchains and its corresponding proof spec required to verify the proofs against the client's consensus state. This process allows transported data packets across the relayers to be verified natively on the corresponding clients on the destination chain. However, running tendermint consensus on-chain is prohibitively expensive (26M gas per non-adjacent header verification on EVM). Therefore, Polymer Lab proposed zkMint as a ZK alternative to Tendermint consensus. With zkMint, Polymer can optimize the ZKP of the block header to any signature scheme and execution environment and reduce header verification cost to 300k gas on EVM. Polymer's solution tradeoffs the generalizability for adaptability of the light client on each chain.

Similar to previous examples, Polymer's security will be reliant on the light client protocols. So it would be based on the chain's security (cosmos and some EVM chains) or the sync committee (Ethereum). However, there is also a scenario where Polymer sits on a highly modularized chain, e.g. Sovereign rollups on Celestia. In such a case, a single light client that is used to perform verification of the block header from a monolithic chain will split into modularized counterparties as shown below.

While modularity comes with the advantages of lower cost and larger throughput, such as with external DA, it also comes with the disadvantages of fragmented securities that could be further discounted depending on how light clients interact with the security layer, e.g. light nodes need to perform secure DAS on the DA nodes.

zkBridge security = Light client security != underlying chain security

Many zkBridges claim to be trust-minimized and inherit the securities of the underlying chains. But you might notice by now

that all of the zkBridges/zkIBC have a heavy reliance on light clients for consensus proof. While some chains provide full validator set access to their light clients to allow full validator signature verifications, Ethereum doesn't. This presents a very different picture of security around Ethereum light clients, and I would like to discuss whether using such light clients is really trust-minimized (It's NOT).

Security of light clients

As mentioned, light clients on many chains have full validator set access, but Ethereum light clients and a few others do not due to the "heavy" compute overhead from verifying the signatures of more than half a million validators. Succinct Labs has previously performed analysis on the likelihood of colluding sync committee. They have suggested that the sync committee has nearly ~0% (1.72*10^-34 chance) of collusion under the condition that ⅓ of the validator are honest and requires 90% of the sync committee to agree for a valid block result. Although the sync committee "seems" to offer few-of-N trust from that analysis, I believe that it does not grant trust-minimized security and the analysis is inconclusive for the following reasons. To start, check out the following statistics on Ethereum full nodes.

1. Node Centralization and Operator Dependence:

Despite Ethereum having over 600k validators, the number of full nodes that are hosting them physically is currently around 8k. Assuming there is a power law distribution on the validator count per node, it will not be surprising to see well-resourced node operators running several thousand validators at once. This also means that it increases the probability that a single entity has supermajority control of the 512 light client validators (just need 342). This skewed distribution of validator across node operators also suggests that validator collusion is not an equally independent occurrence, which is assumed by Succinct's analysis based on binomial probability (assumes independence & same % for all occurrences). I believe the validator collusion is dependent on the colluding node operators, and since each node operator hosts a varied number of validators, it will be inconclusive to determine the probability of the sync committee collusion by framing it as an equally independent occurrence for validator collusion. It would be worthwhile to spend more time calculating the probability of sync committee collusion with the node operator dependence assumption as part of future work.

1. Coordination risk without time constraints:

Probability analysis is an effective measure for the likelihood of system collusion where the collusion via private coordination is improbable, possibly due to time constraints. However, since the sync committee rotation happens every 27 hrs, it is well within the realm of possibility for sync validators to coordinate and collude post-rotation, which renders the probability analysis ineffective in practice.

1. Weak replicability of the consensus:

A few-of-N trust system only needs to trust a few to guarantee that the system is behaving as expected. Such a guarantee comes from the replicability of the system's state where the probability of false replication decreases exponentially with the number of sampling S

. For example, data availability sampling (DAS) allows the replication of the available data via erasure correctingReed-Solomon codes resulting in an exponentially decaying probability of R-S as S

increases (The ratio R= N/K

where N is the N-bit representation of the Reed-Solomon code and K is the K-bits representation of the original data). And with a high enough S, which depends on the configuration of the erasure coding, the original data can be fully reconstructed pseudo-deterministically. However such isn't the case for the sync committee; it will be at best a good approximation unless each validator also enforces the "erasure coding" type of behaviour where a validator holds attestation of N' other validators for redundancy, which enables replication of the consensus via random sampling.

1. Incentive alignment:

While there is a sync committee reward to the partaking validators, there are no penalties for their misbehaviours. Furthermore, the sync committee is picked randomly, so there is no consensus and no added incentives on safeguarding the selection of the committee.

For the reasons above, I argue that Ethereum light clients cannot be considered trust-minimized, and it is wise for more middleware relying on such light clients for consensus proof to migrate to mechanisms that allow verification of the full Ethereum consensus that ACTUALLY provide trust-minimized security (hopefully 1-of-N)

.

Verifying the full Ethereum Consensus

To fully verify the consensus, we need to verify the signatures of all the attesting validators of a block, which is about ~600K signatures as of May 2023. Naively generating the proofs and verifying all the signatures would be prohibitively expensive,

and the verification process also needs to be cryptographically verifiable to achieve minimal trust overhead (ideally 1-of-N security). Currently, there are four approaches that can be explored in achieving trust-minimized full consensus verification, all of them are variations of ZK light clients:

1. SNARK/STARK recursive composition approach
2. Random Sampling approach (extension of sync committee)
3. Commitment-based approach
4. Folding-based approach

Let's dive into how each of them works and discuss their trust-assumptions/tradeoffs.

SNARK/STARK recursive composition approach:

As previously mentioned, verifying the BLS signature on-chain can be expensive, therefore it is necessary to use SNARK to perform abstraction to reduce the gas cost to cheaper verification of the proof. However, performing proof generations for ~600K validator signatures will most likely take hours to complete. Therefore, having a highly optimised proving scheme would be necessary. For example, using Plonky2 with Starky could allow many proofs to be composed recursively with parallelized compute over the proof generation for all signatures, which would look something like the diagram below where you end up with each layer of parallelised prove generation and single SNARK at the end.

The advantage of using STARK, like Starky, on the first layer of signature verification is that STARK generally offers a faster proving time, as shown in the graph below, at the cost of a larger proof size. The proof size won't matter much in this case since Plonky2 can recursively compose the proof resulting in a smaller footprint. Another advantage of using Plonky2 is the ability to adjust the proof size based on the required proving time. Small proof size matters for cheap on-chain verification, but all the composite Plonky2 SNARKs that are generated during the recursion process do not need to have small proof, as long as the memory allows. Since Plonky2 is built with FRI, an LDT typically used for STARK, it enables a variable degree of proving speed such that:

- faster proving comes with larger proof
- slower proving comes with smaller proof

This recursive composition schema takes advantage of such property and optimizes the recursion speed for signature verifications.

A concern with this approach would be the latency. Even with parallelized and optimized recursion, the total proving time may take quite some time (proving for 512 sync committee signatures may take up to 30 mins). Furthermore, a non-plonky2+Starky implementation of such recursion may involve non-native field arithmetic which introduces additional performance overhead from emulated arithmetic. Cyclical pairing groups for the polynomial commitment scheme of SNARK could potentially help reduce the overhead but more benchmarking is required to have a better idea of its performance.

Succinct Lab has also proposed a solution where the BLS signature and SHA256 can be verified within their STARK-based VM in Plonky2. While details are not out yet, I believe a customised zkVM would provide them with better generalizability in performing recursive proof for various signatures and hash functions instead of having many customized circuits.

Random Sampling Approach:

Similar to the sync committee, the protocol will randomly select the validator signatures, based on RANDAO, to recursively prove and verify. This approach is also an approximation of the full consensus but it provides more flexibility to the tradeoff between cost and security. Given that the number of sampled validator signatures can adjust, users who require a higher level of security can choose to bear the cost of sampling more validators while those who do not need strong security can pay less by sampling less. This essentially gets rid of the 512 validator constraints and leaves the security and economic decisions to the protocol/users.

Commitment-based approach:

Instead of individually verifying each validator's signature, this approach allows the prover to demonstrate that the signers of the signatures S form a subset of the entire validator set T. It then becomes possible to efficiently verify the resulting aggregate BLS signature. The prover achieves this by using the aggregate BLS signature of the signers S, the corresponding aggregated public key (APK) of S, and the public keys of all validators T to generate a SNARK proof, showing that S is a subset of T. The verifier needs to maintain an up-to-date KZG commitment, where the prover computes the proof for the Merkle commitment of block roots to T and the corresponding KZG proof for the update. This commitment represents the entire set of BLS public keys of validators T, without storing any of the actual public keys. The verifier can then efficiently verify that the committed validators T correspond to the signed validators S through the SNARK proof. It's important to note that in Ethereum's Casper FFG, the attestation vote doesn't reach a supermajority vote until the block is published. Therefore, multiple rounds of aggregate BLS signatures, involving different sets of validators from attestation committees, need to be batched together for the SNARK proof to sufficiently demonstrate a supermajority vote on the consensus. To help visualise the process, a diagram illustrating the flow of signature verification is provided below (a bit messy but I tried

lol).

Commitment-based approach has the advantage of requiring less computation for proving, which can be further reduced via batching of the aggregate BLS signatures. Polytope Lab has proposed this approach based on the paper released by Web3 Foundation on accountable light clients. This is currently being built on a Polkadot parachain that comes equipped with precompiles for the required curve for aggregate BLS signature verification such that it can perform verification natively. However, this could also be done on Ethereum by SNARKing the aggregate signature, either recursively or via folding.

Folding-based approach:

A folding scheme allows the prover to compress two instances of witnesses (some inputs) into one such that ZKP can later prove the validity of both at once. The folding process needs to be verifiable via augmented arithmetization schema such as augmented R1CS, which performs simple verification of the N-1 instance of witness with respect to N. Here is a high-level workflow for the folding of two input sets A and B.

There are a few different folding scheme options, such as Nova (single-function folding), Supernova (multi-function folding), and Sangaria (Nova for Plonk arithmetization). But for consensus verification, which involves repetitive execution of a single BLS signature verification function, it may suffice to use Nova or Sangaria.

Now, It should be clear why folding can be super powerful for signature verifications. Thousands of signature inputs can be folded and made into a Merkle tree to

1. reduce the compute overhead from proving SNARK/STARK at each step of aggregation
2. and parallelize the folding steps.

At the end of this process, a single folded representation of the full validator signatures remains to be proven and verified via SNARK/STARK.

While most of the folding schemes are not yet battle-tested and may still have a big room for performance optimization, folding-based approaches could be the most performant, flexible, and scalable method of consensus verification long term. There aren't many projects working on this yet, but I believe this is a promising direction that many will explore its application in creating an efficient ZK light client.

Cost of trust-minimized ZK light client

I have discussed four non-mutually exclusive options to build a ZK light client to perform full consensus verification. Most of them provide more trust-minimized security than existing Ethereum light clients. However, it is important to note their drawbacks.

1. Trusted-liveness for parallelized compute: Proving scheme that requires concurrent node operations, such as for parallelized proving and folding, would need a strong liveness guarantee from the underlying provers and node operators. If one operator performs a griefing attack during the proving procedure, it could halt or delay the proof generation process resulting in a temporary loss of liveness of the light clients. In the event of such an attack, it only needs at least one honest prover to perform all the computations to maintain the liveness, but that would not reap the benefit of parallelized computing and can result in a long delay, which in effect could be a censorship attack. To alleviate such delay and temporary loss of liveness, the ZK light clients could adopt redundant provers to increase liveness guarantees at the cost of user expense and wasted computing.
2. High Compute Cost: There will be added compute costs for proof generation. Depending on the proof scheme, this cost could be prohibitively expensive; running many provers in parallel is a big expense. And as mentioned in the previous point, adding redundancy to the prover set could incur additional compute costs.

To better compare and summarize some of the points mentioned previously, here is a table for comparison across different consensus light clients.

Consensus verification != State Validity

Besides consensus verification, there is also another important aspect of the blockchain that is worth mentioning in the context of bridging: State validity, the idea that the transactions and their blocks are valid. Consensus light clients, including ZK light clients, infer the state validity by making sure that the majority of validators agree to the proposed block, but they do not have a direct view of the validity of the block. This nuance is important in some edge cases. Let's imagine a scenario where Ethereum is 51% attacked and invalid blocks are proposed. If dApps were only relying on ZK light clients, they will not catch any signs of attack because newly proposed blocks receive over majority attestation. However, if the dApps are also monitoring the state validity potentially via full node, they will be able to detect and reject invalid blocks and prevent any malicious transactions. To achieve this with ZK light client is improbable due to the amount of block data that needs to be downloaded to perform the re-execution of transactions, which essentially makes it a full node. One possible workaround to access state validity in a trust-minimized manner is to rely on a verifiable full node whose execution trace can be recorded

and proven via ZKP. And state validity proof could be combined with consensus proof to generate proof that verifies both the consensus and the state validity. This would be a similar approach to developing a ZK-Archival node for trustless data retrieval, except specifically for state validity in this case. Even relying on a single verifiable full node can potentially help avoid damages from invalid blocks, and it could run in parallel to ZK light clients as an added security measure. The feasibility of a verifiable full node requires further deliberation, but using a zkVM like RiscZero and taking “images” of each execution step of the full node could be one possible way to achieve a verifiable full node. With RiscZero, state validity proof can be generated with 1-of-N security with 1-of-N liveness, assuming there is a distributed set of provers performing compute.

Concluding thoughts on consensus verification and trust-minimised bridging

The recursive composition approach, random sampling approach, and commitment-based approach would be feasible within the short to medium-term future, many teams like Jump, Succinct, Polytope, NiL and HyperOracle are already working on it. And the folding-based approach could be the end goal for providing both superior performance, security, and flexibility. Having said that, there are varieties of contexts requiring consensus verification for different purposes with varied levels of security guarantees, such as bridging small vs large amounts of assets. Different approaches offer different sets of tradeoffs between performances and security assumptions. Folding-based approaches seem to theoretically offer the least tradeoff so far, but we shall see how it actually performs in practice.

The emergence of trust-minimised ZK light clients provides a more secure consensus verification, which enables more trust-minimized bridging. However, existing bridges that have modular designs could adopt such light clients to bolster their own security. Many of the existing bridges like LayerZero, Wormhole, Celer, and Axelar have already garnered significant liquidity and ecosystem moat and are at a big advantage long term as they also start adopting ZK light clients. For example, the Wormhole team has always had the vision to adopt a more secure option for consensus verification, hence built Wormhole with modularity in mind. This will enable them to transition to trust-minimised security by integrating with a ZK light client once the tech is more battle-tested. It will be crucial for newer zkBridges to aggressively develop their liquidity and ecosystem moat while winning the technical race.

Besides consensus verification, there is also an important consideration of state validity proved via verifiable full nodes using zkVM. I believe this is another interesting future research direction that should deserve more attention.

I haven’t had the chance to discuss many other important areas of trust-minimized applications including verifiable computing and data. I believe they could enable many trust-minimized use cases on-chain that wouldn’t be possible otherwise. I will further elaborate on those in the next part of the article for The World of “Trust-minimized” Middleware.

MINMAX