Summary

I propose a cooperative proving network with minimal alterations to Fernet. Provers are randomly from the stake table using a VRF. The proving network is a series of redundant binary trees that build the root proof in parallel. Provers inside the binary tree send their proof chunks across trees to compensate for Byzantine provers. The proving network can be parameterized for f

faulty provers out of n

total provers. If the sequencer does not distribute the data to provers in a timely manner, the provers for a timeout certificate to alert the rest of the network to stop building off that particular block.

Goals

1. Create a Byzantine fault tolerant proving mechanism that, under reasonable assumptions of synchrony and an honest sequencer, will always produce a valid block proof
2. Never slash for liveness faults
3. Maximize fault tolerance and parallel proving through decentralization of provers
4. Maintain decentralization by 1. keeping the barrier of entry for provers low, particularly as proving becomes more computationally efficient and more accessible on commodity hardware, and 2. minimizing incentives to create the fastest proof by instead focusing on proofs that are created “fast-enough”
5. Optimistically pipeline proving-related network messages such that they add minimal network overhead to theFernet protocol.

Design

Model and Participation

Some number of n

provers are elected to prove some block B

. The protocol can tolerate up to f

of these provers being Byzantine. Each prover is at minimum responsible for proving num_leaves / n

proof chunks. Many provers will additionally be chosen to prove internal nodes of the proving tree.

For each block, n

provers are elected from the stake table to participate in the proving protocol for that block. n

can be a reasonable number, chosen through network governance, to optimize between decentralization and network overhead. It can be updated at epoch boundaries to adapt to network needs. Prover election should be unpredictable and randomly distributed to ensure maximum fault tolerance.

Provers can be elected as in Fernet using the block number and RANDAO value as inputs into a VRF. Provers must have a deterministic ordering so that we can assign proving chunks, but this ordering can be revealed and verified at a later time [1]

.

As with Fernet, the liveness of this protocol is dependent on the liveness of the L1 and DA layer. We assume a partial synchrony model.

Each phase of Fernet will require a timeout window by which that phase’s data must be posted to the L1. This can be calculated in terms of L1 blocks instead of real-world time and should be set conservatively. The protocol can optimistically proceed faster than this window.

It is worth noting that since we want to maximize parallel proving, we want unique provers at each lateral of the binary tree. Thus our security model tolerates up to f

faulty provers

, not f

faulty units of stake. Stake does not necessarily correlate to proving power, and we should be cautious how this changes our

security assumptions of the network.

Protocol

Prior to proving

In parallel to the proposal phase the proving network is elected as in Fernet using the block number and RANDAO value as inputs into a VRF [1:1]

.

In parallel to the reveal phase two operations happen: 1. the sequencer for block B

sends the designated proving data to prover leaf nodes over direct, point-to-point network connections, and 2. provers begin to publish their network information and verifiable ordering position to a DHT. Internal tree provers can optionally choose to wait to publish their identity info until the network reasonably needs them to do so [2]

. Provers can either optimistically begin proving once they receive their data or wait to verify that the block data was correctly certified the DA layer. For the sequencer, the reveal phase only requires their participation in the beginning of the phase when they send data to the DA layer. The sequencer is idle while it waits for the DA layer to confirm the data. The sequencer can make use of this idle period to disseminate the proving data to provers. A key point here is that setup of the proving network can be entirely pipelined

with the existing Fernet protocol. This proposal does not add any additional waiting time to Fernet.

Provers will monitor the DA layer for a certificate of availability. If a prover receives its data before some time window delta

after the DA certificate is published, it proceeds with the rest of the protocol. If the prover does not receive the data within the time window, they send a timeout vote to a VRF-elected leader among the provers.

If this leader collects f+1

timeout votes it forms a timeout certificate[3]

. It posts this certificate to the L1. This serves as proof that future sequencers and provers should stop build off this block since it will never be proved. This optimization isn't strictly required, but it helps the network recover more quickly if the sequencer does not release the proving data in a timely manner. If provers have sent timeout votes but do not see a timeout certificate posted to the L1 within some additional time window, they then send another timeout vote to the next VRF-elected leader. They do this until either the proving window has elapsed, they see a timeout certificate, they see a proof for the block, or they have tried f + 1 leaders [4]

.

It is possible that the sequencer did not send the data in the time to have the network prove it, but that the network is asynchronous and a timeout certificate can't be formed. This doesn't violate safety or liveness. Instead the protocol will simply have to wait for the entire proving window to time out, potentially causing a larger reorg. Reorgs work the same way as described in Fernet.

During proving

Provers are assigned unique nodes in each of c

proving trees where c

is defined such that (f / n)^c < tolerance

. For maximum fault tolerance c=f

, but maintaining f

proving trees is redundant and wasteful for even moderate sizes of n

. Therefore we use a probabilistic tolerance that represents the chance of all root proving nodes being Byzantine. So long as provers are randomly and unpredictably chosen, we can assume that Byzantine provers have an f / n

chance of being a root note for any given tree[5]

.

For maximum fault tolerance, the ordering of provers at each lateral in a proving tree should be unique from other proving laterals in other trees. This ensures a single Byzantine prover does not end up in the same position across several different

trees. Proving work should also be equally distributed among provers within a single tree lateral such that each prover is responsible for num_nodes_at_lateral / n

chunks. We can probabilistically approximate this property by randomly assigning provers to nodes in the tree.

Inspired by an inverted version of Turbine and DHT-striping techniques, provers send their outputs to their direct parent and

all of their would-be parents across each proving tree. This allows proving upwards in the tree to continue even in the presence of Byzantine internal nodes.

Provers do not need to maintain any long-term network connections to other provers. Child provers can initiate a one-time connection to send proofs to their parent provers. Provers only need to connect to their parents and listen for incoming connections from their children. All network identification information can be stored in a DHT for lookup. This reduces network overhead drastically compared to a broadcast or gossip network [6]

.

Finally, starting at the top-most m

level of the proving tree (where m

is very small, such as 2) provers also send their output to the sequencer. This allows the sequencer to optionally finish up the last few steps of proving should all c

root nodes be Byzantine.

After Proving

Each of the c

root proving nodes will submit their final proof to the L1. The sequencer may optionally submit their proof as well. As in Fernet, the first proof to make it to the L1 within the proving window finalizes block B

. The other proofs are only submitted to be eligible for rewards. They are not required for finality.

The diagram below explains the full protocol [7]

[

Aztec.drawio

1323×781 42.3 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/a27511d1c6ce49d2a94b2afccff617b0447268b5.png)

Incentives

Each block will reward sequencers and provers separately. The reward per prover is proportional to the amount of proving they did for a block, which is the number of chunks they proved across all final proofs that were submitted. Similar to the proof of stealing prevention mechanism from Staking Proving Network for Fernet, proof chunks can be signed by their provers. A root proof will only be valid if it contains proof of participation from the correct provers. Correct provers can be verified using the VRF output. All root proofs that are posted to the L1 within the allotted time window are rewarded.

Sequencers are not rewarded for any proving they do unless they have been legitimately elected as a prover in the protocol.

In this proposal, provers and sequencers are only slashed for attributable

invalid behavior, such as creating incorrect proofs. They are never slashed for liveness failures.

Reasoning

1. This proposal elects provers from the stake table for the reasons outlined inCooperative proving network for Fernet and Staking Proving Network for Fernet.
2. Rewarding all proofs submitting within the proving window 2. encourages decentralization and 2. lowers the barrier of entry. This design allows more commodity level hardware to participate in the proving network (particularly as there are more advances in hardware / techniques). This proposal only cares that proofs are generated “fast enough,” and purposefully removes competition for parties to be the quickest prover. Competition to be the quickest prover pressures only centralized, highly capitalized parties to participate.

Comparisons

- Sidecar and Fernet on the rocks - These proposals take an opposite approach by using a competitive proving market rather than a cooperative one. It is possible to combine these proposals with this one, however. During the proposal phase sequencers can signal whether or not they wish to use the proving network or they own proof generation mechanism. They can use either Sidecar or Fernet on the rocks to facilitate their own proof generation. However the sequencer will receive no extra rewards for choosing their own proof generation. The rewards that would have been distributed to the proving network will be burned instead due to the reasoning explained above. Lastly, these two proposals abstract away a lot of complexity from the Aztec protocol, making them realistically much easier to implement. Oftentimes the simpler approach is better.
- Cooperative proving network for Fernet and Staking Proving Network for Fernet - These proposals are quite similar to this one. The main difference is that this proposal takes a different stance on liveness slashing and rewards distributions.

Known Limitations / Open Questions

- Rewarding all proofs posted within the proving window is very costly if we want the rewards to reflect the resources required to created the proofs. This is particularly true if these rewards can't be funded through slashed activities like in other proposals. However, I'd argue using slashed funds to fund rewards creates incentives to DoS slashable parties, which is one reasons I strongly discourage slashing for liveness failures.
- There is no reputation system. Although provers will receive no rewards if they can't prove within a reasonable time, this protocol makes no effort to choose reliable or "good" provers. However, reputation systems must be carefully designed in order to not increase the barrier of entry for new participants and centralize the participant pool to only publicly known entities with public reputations.
- See note [1:2]

.

1. We want something with these properties: 1. the identities of provers are kept secret until provers choose to reveal them, 2. we can ensure a deterministic ordering without needing provers to reveal their identities to create the ordering. I need to think more about what this cryptographic construction could look like. In the naive case, we can forgo prover privacy and have the VRF calculated over provers' public keys instead of a SNARK of their private key. This could potentially open the protocol up to DoS attacks of provers. However an attacker could arguably attack the sequencer (which must be known and actively participate in the protocol) to stall the network anyway. So perhaps this DoS vector isn't urgent to address ↩ ↩ ↩
2. For example, the root prover may choose to wait to publish its identity information until 70% into the expected proof time. Provers must know who their parents are in the proving tree, however. Also note that a single prover key can be associated with an arbitrary number of network identities. ↩
3. This f

parameter can be different than the f

parameter described under the Model and Participation section. Furthermore, we only need f + 1 timeout votes for a timeouts certificate to ensure that at least 1 honest node correctly voted. In this case conflicting, valid certificates do not affect the safety or liveness of the protocol, so we do not strictly need the usual 2f + 1

votes to form a certificate nor do we need to constrain f <= n / 3 + 1

. (I'd like to do a proof to be fully confident in this statement)↩

1. Note here that we are 1. not using a full consensus protocol, and 2. we are not affirmatively voting on the timeliness of data. We can achieve property 1 because we a. do not slash for liveness failures, and b. because the timeout certificate is only an optimization, not a safety or liveness requirement. Thus we can get away with a much simpler best-effort broadcast protocol instead of a full consensus protocol.

↩

1. This is a bit of a an overly idealistic assumption to make.↩
2. Empirical benchmarks would be beneficial for analyzing this design, but that is outside the scope of this proposal.↩
3. Many claims in this proposal need proofs. I plan to work on that in the future. ↩