Cooperative proving network for Fernet

Attempts at designing a prover network that plays nicely withFernet.

A cooperative proving network

We'll shoot for a cooperative proving network, as opposed to a competitive one. Similar to sequencers, provers are required to stake to participate in the protocol. On every block, for every section of the proof tree, a VRF is evaluated using the prover's key. This VRF outputs a score that ranks each prover for working on a specific part of the proof for a specific block.

Fee split and burn

We first adjust Fernet's sequencer scoring function by multiplying the resulting VRF score by a fee split percentage. During the proposal phase, sequencers also submit a proposed FEE_SPLIT

(from 0 to 1), and their resulting score is VRF_OUTPUT * (1 + FEE_SPLIT)

. Should their block be elected, the captured fees in that block plus any constant block rewards are split according to that value between the sequencer and the proving network. This incentivises sequencers to forward as much value as they can, while remaining competitive. Note that all captured MEV remains with the sequencers, though this could be revised by allowing the FEE_SPLIT

to be greater than 1.

Similarly, each prover has a VRF output score that determines their eligibility to build a section of the proof tree. This score is further adjusted by a FEE_BURN

value (from 0 to 1), so their resulting score is VRF_OUTPUT * (1 + FEE_BURN)

. Should their proof be included in the tree, they get FEE_BURN

of their respective reward, and the rest is burned. Allowing values greater than 1 doesn't make sense here, since provers don't have incentives for doing the work other than the reward itself.

VRF considerations

The VRF can be evaluated on the provers public or private key. Using the public key lets anyone know the complete prover ranking, potentially leading to less wasted effort. It also allows the lead to set a precise FEE_BURN

that allows them to keep the leading position. On the other hand, using the private key allows provers to retain privacy, and it may also lead to higher wasted effort which is not necessarily bad and could provide good redundancy. This same discussion applies to the sequencer VRF.

As for the randomness source, we could use the same RANDAO value as the sequencer, combined with their VRF output. We rule out using the block hash, since sequencers could manipulate it to ensure that provers they control end up with a higher score. And we don't use the raw RANDAO value but instead we mix it with the sequencer VRF output, so that different block proposals from different sequencers lead to different prover selections.

Prover selection and rewards

For each node in the proof tree, up to PROOFS_PER_NODE = 3

proofs are expected to be selected. Each prover that gets their proof included in a finalised tree with NUM_NODES

gets a reward equal to:

PROVER_NETWORK_REWARD = (TOTAL_TX_FEES + BLOCK_REWARD) * FEE_SPLIT PROVER_REWARD(i) = PROVER_NETWORK_REWARD * (1 - FEE_BURN(i)) / (NUM_NODES * PROOFS_PER_NODE)

Example of rewards distribution

Total block rewards & fees: 1000 AZT

Number of transactions in the block: 32 (ie 31 nodes in the proof tree)

Sequencer FEE_SPLIT

: 80%

Total rewards to provers: 1000 AZT * 80% = 800 AZT

Maximum individual reward per prover: 800 AZT / (31 * 3) = 8.6 AZT

Effective reward for prover with FEE_BURN=40%

: 8.6 AZT * 60% = 5.16 AZT

Should there be competing roots submitted, the winner is the one with the highest total score summed across all its included proofs. Remember that the score of a prover for a given node is determined by the VRF and boosted by their FEE_BURN

.

This incentivises provers to build upon proofs with the highest scores, but grants them flexibility to choose others if those proofs are not shared. Furthermore, provers higher up in the tree are encouraged to include all 3 proofs for each subtree, in order to increase their score. If less than 3 proofs are included for a given node, the fees that correspond to the missing ones are burned.

Example of proof tree scoring

Let’s assume a proof tree of size 3, with two base rollups and a root rollup. On the left tree, prover R1 (who has an effective score of 60) includes the proofs from provers G1, G2, G3 for the green node, and from B1 and B2 for the blue node. On the right tree, prover R2 (with a score of 40) chooses the same G1, G2, G3, but adds the proof from B3 for the blue node. Thus, the total score for the right tree is 10 more than the left one, and wins.

[

image

1185×472 97.2 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/d4a568a4f44bca95ac39d6fd46b84e259ae42e50.jpeg)

Note that it is possible that a prover has the highest score for a node and is still not included. However, this would be the result of all 3 provers for its parent node ignoring it and choosing other proofs instead (at the risk of reducing their own score), making it unlikely.

Statefulness

Allowing provers to be stateless is only possible if there is another entity incentivised to push them whatever state they need to execute the proofs. In this model, the sequencer cannot be trusted to fulfil this role, since this grants the sequencer power to choose whatever provers they want (by pushing data only to them), defeating the purpose of a protocol prover selection algorithm.

This seems to lead to provers being stateful. However, this could also lead to prover-PBS (?), in which the entity that is selected to propose a proof and the one that actually builds it is not the same. Staked provers could just be regular nodes that follow the chain, and if they are selected as provers, they could offload the computation, along with all necessary inputs, to a specialised proof builder such as nil.

It’s also worth noting that we could remove the statefulness requirement for merge rollup provers. If a merge rollup prover trusts the input from its children, which is a big assumption, then it can assemble the merge proof without any state. If we wanted to exploit this, we could allow provers to sign up separately as base and merge provers.