Update: The latest version of this proposal can be foundhere.

Fair Election Randomized Natively on Ethereum Trustlessly

(Fernet

) is similar to Whisk-y, Schmequencer, and Irish Coffee in that it relies on randomness for sequencer selection. However, it doesn't select a fixed set of leaders, but assigns a weight to each of them to secretly rank them. Whether a sequencer chooses to propose a block or not depends on their expected rank.

This proposal also draws from B52 in that there's a strong separation between sequencing and proving. Since sequencing is cheap, we can deal with duplicated efforts. And once tx ordering is set, then the proving work is well-defined. This purposefully removes one of the sequencer responsibilities requested in the RFP: Build a rollup proof from completed proofs in the proof pool

.

Building blocks

VRF

: We first define a VRF over the RANDAO beacon bound to the sequencer keypair, such asa BLS signature over the RANDAO value, or a small SNARK as suggested in Irish Coffee. This gives us a secret, random, verifiable value, bound to each sequencer, that updates with every Ethereum block.

Staking contract

: We keep a staking contract on L1, where sequencers can register themselves with a public key, and stake a fixed amount of funds to participate. We introduce an unstaking delay to allow for slashing dishonest behaviour.

Rollup contract

: A rollup contract on L1 that tracks the sequence of txs and the proofs. Note that these will be submitted separately.

Protocol

- Sequencer submits order

: On each L1 block, any sequencer can submit a tx ordering to the L1 rollup contract. We call this an "unproven block". The data published is an ordered list of L2 tx hashes, an identifier (the hash of the list), the identifier of the previous block that this one builds upon, and the sequencer VRF output. * The unproven block could also include the tx's payload. See the questions section for tradeoffs on this decision.

- The contract needs to store only the identifiers; the raw list of txs can be kept in calldata, but it's important that it's made public at this stage.
- In case there are multiple submissions, the contract will keep track of all of them. However, the protocol considers the canonical one as the one where the VRF output is closest to the current RANDAO value

, among all the valid ones.

- An unproven block is considered valid

if all its published tx hashes correspond to valid L2 txs, they are not repeated, their ordering is correct, and its previous block is canonical

. This notion of validity cannot be enforced by the L1 rollup contract, but can be easily verified by an L2 node.

- Once the L1 block is mined, any L2 full node can pick the canonical unproven block, check whether it is valid, and execute the state transition function. This provides an early optimistic finality for L2 txs

.

- If no valid unproven blocks are published in block N, then in block N+1 sequencers should reference the canonical unproven block from block N-1.
- The unproven block could also include the tx's payload. See the questions section for tradeoffs on this decision.
- The contract needs to store only the identifiers; the raw list of txs can be kept in calldata, but it's important that it's

made public at this stage.

- In case there are multiple submissions, the contract will keep track of all of them. However, the protocol considers the canonical one as the one where the VRF output is closest to the current RANDAO value

, among all the valid ones.

- An unproven block is considered valid

if all its published tx hashes correspond to valid L2 txs, they are not repeated, their ordering is correct, and its previous block is canonical

. This notion of validity cannot be enforced by the L1 rollup contract, but can be easily verified by an L2 node.

- Once the L1 block is mined, any L2 full node can pick the canonical unproven block, check whether it is valid, and execute the state transition function. This provides an early optimistic finality for L2 txs

.

- If no valid unproven blocks are published in block N, then in block N+1 sequencers should reference the canonical unproven block from block N-1.

- Proving starts

: Every L1_BLOCKS_PER_ROLLUP

L1 blocks, provers pick the canonical chain of unproven blocks since the last proof, and start proving. * We can set L1_BLOCKS_PER_ROLLUP

to 1 if we want a proof for every block. However, due to the high cost of verifying proofs on-chain, it may make sense to batch a few blocks. For instance, we could set L1_BLOCKS_PER_ROLLUP

to align with Ethereum epochs.

- How provers are selected is beyond this proposal, but we could favor a subset of provers to reduce duplicate work. For instance, the reward that a prover gets could be proportional to 1) being selected in a committee using a strategy similar to Ethereum's, 2) the inclusion delay, so other provers can jump in if the selected ones take too long, and 3) the rewards of the proofs being aggregated.

- We can set L1_BLOCKS_PER_ROLLUP

to 1 if we want a proof for every block. However, due to the high cost of verifying proofs on-chain, it may make sense to batch a few blocks. For instance, we could set L1_BLOCKS_PER_ROLLUP

to align with Ethereum epochs.

- How provers are selected is beyond this proposal, but we could favor a subset of provers to reduce duplicate work. For instance, the reward that a prover gets could be proportional to 1) being selected in a committee using a strategy similar to Ethereum's, 2) the inclusion delay, so other provers can jump in if the selected ones take too long, and 3) the rewards of the proofs being aggregated.

- Proof is submitted

: Eventually the provers submit their proof for the canonical block, which becomes a "proven block". We now know that this block is valid, but it may not be the canonical, since provers could submit proofs for any of the valid blocks. * Again, anyone may submit a proof for the canonical block, and selecting a winner is out of the scope of this proposal. Still, following the idea from the point above, we could choose the canonical proof as the one with the highest accumulated rewards.

- If the rollup contract receives proofs for more than one valid block, then it can decide which one should be the canonical block by checking the stored VRF outputs. We could slash provers for submitting a proof for a non-canonical block.

- Again, anyone may submit a proof for the canonical block, and selecting a winner is out of the scope of this proposal. Still, following the idea from the point above, we could choose the canonical proof as the one with the highest accumulated rewards.

- If the rollup contract receives proofs for more than one valid block, then it can decide which one should be the canonical block by checking the stored VRF outputs. We could slash provers for submitting a proof for a non-canonical block.

- Proof is final

: After N blocks, we assume that honest provers have had the time to submit proofs for the canonical block, so we consider the current proven block as “finalized”. * Only at this point are rewards paid out.

- It may be the case that the canonical block didn’t get any proof submitted, but other valid ones did. If this happens, to ensure liveness, the protocol picks the current finalized block is the canonical one.
- Only at this point are rewards paid out.
- It may be the case that the canonical block didn’t get any proof submitted, but other valid ones did. If this happens, to ensure liveness, the protocol picks the current finalized block is the canonical one.

Design decisions

- Not enshrining MEV at the protocol layer

. Unlike B52, this design leaves the door open for hybrid PBS as works on Ethereum today: proposers are selected at random, and pick bids from builders from an off-protocol marketplace. While there are currently pushes for moving PBS to the protocol layer (thanks Joe for sharing that article!), I personally think that this is still an immature area of research and subject to frequent changes. I don’t think we can push research forward fast enough to come up with a secure design, much less implementation, within 6 months. Furthermore, having PBS as a sidecar allows us for rapid iteration on it, and gives the community the flexibility to outright remove it in the event of a critical bug without requiring a major fork.

- Splitting sequencing and proving

. While Goblin and Ultrahonk should deliver major improvements in proving times, at the moment we are looking at a proving time of 2-10 minutes per block (and that’s assuming maximum parallelization). Having an early checkpointing mechanism allows us to define the tx ordering quickly, which is the only indeterministic part of block-building (aside from who reaps the proving rewards), so clients can get quicker finality. It also leads to less wasted efforts: submitting a tx ordering that isn’t chosen as canonical is not a big loss, but submitting a proven block that doesn’t get picked up means significant wasted computing power.

- Ranking over choosing leaders

. Proposals like Schmequencer or Irish Coffee rely on choosing one or a specific set of leaders. This requires careful modeling (and potentially tweaking via protocol upgrades) to nail how many leaders would be probabilistically chosen. Here, having a ranking function for anyone who wants to participate allows for sequencers to quickly iterate different strategies. Furthermore, by splitting sequencing and proving, attempting to participate in a block is relatively cheap.

- Single-block leaders

. Unlike Irish Coffee, this design gives control to a sequencer for a single block. This prevents monopolization of the chain for long (as in 10 minutes) periods of time, which could lead to oracle manipulation attacks for example.

- Requiring staking for participation

. Unlike Schmequencer, we require sequencers to explicitly stake in the L1 for selection. This keeps the entire selection algorithm on L1, so early issues on L2 would not impact it. It also prevents accidental delegation of selection power: eg if I put my native token in a 4626-like vault, I’m unwittingly giving more chances to that vault to be selected as the next sequencer. Last, staking leaves the door open to implementing slashing (even social slashing) in the event of detecting malicious sequencer behaviours.

Open questions

- Can we get a better backronym?

I had to ask ChatGPT for help on the T. I originally had used “ta-da” which was fun but a bit too much. Still, I feel there’s room for improvement there.

- What to stake?

While we probably want to require a fixed amount (like Ethereum does with its 32ETH), the question remains of which currency to use. Options seem to be ETH, rETH, or a network native token. I understand that using a native token provides some additional defense against takeovers: if an entity purchases enough native token to take over the protocol, then the token value would drop to zero as the protocol utility disappears due to the takeover, and the unstaking delay means the entity cannot dump their tokens before they become worthless. But economic incentives are not my strong point.

- How to slash?

Related to the above, there are no clearly dishonest behaviours from a sequencer that could be automated for slashing. We

could have governance-powered slashing though, so we can have the community observe and decide to punish behaviours.

- What VRF to use?

This seems to be a recurring question across proposals, so I'll leave it to the experts. Goal is to get something simple and cheap to verify in the EVM.

- What data should be included in an unproven block?

If the unproven block only includes the tx hashes, there's an attack where a sequencer could withhold one of the txs, preventing the prover network from being able to generate a proof for it, and reselling block-proving rights (or grieving provers by waiting for them to submit a proof for a non-canonical block, and then slashing). This can be mitigated by posting all tx data onchain, but this can quickly get prohibitively expensive, since each tx includes a ZKP which is not particularly small. We could look into alternative DA solutions, 4844 included, to solve this.