After settling on Fernet as sequencing algorithm, we attempted to simplify it as much as possible to reduce engineering efforts and get to an early initial working version that we could start playing with. This post covers some of the tradeoffs we are evaluating, along with some open problems we are working on and their mitigations!

Tweaks

What we propose to change from the base version linked above.

Removing build-ahead

To remove the risk of L2-reorgs, we propose removing build-ahead for an initial release. This means that the phases for block N+1 would begin once block N has been finalized, not once it has been revealed. This removes reorgs caused by proof withholding attacks.

Note that this turns Fernet into pretty much Sequencer Schmequencer. Should we switch to that name?

Skipping rollup proof verification

Instead of implementing proof batching, we still submit a proof per block, but do not verify it in order to save gas. This guarantees L2 full nodes have access to proofs, so they can verify that the block revealed by the sequencer is valid. Then, to get finalization on L1, every N blocks we aggregate these proofs into a single batch proof and verify it. Note that whether we move forward with this option depends on more analysis on costs. Thanks to the Scroll team for suggesting this one!

Prover commitment

Assuming an out-of-protocol proving scheme, which is the simplest option to begin with, we propose a prover commitment for coordination between sequencer and prover. Once the sequencer has chosen a proving entity (whether it's an individual, a marketplace, or a network), this entity locks down a certain amount of value as a commitment to generating the proof. This should act as a signal to the network that the proposal from the sequencer is legit, and add more economic security to the finalization of the block.

Sequencer bid

We propose sequencers to be allowed to bid during the proposal phase, as a means to providing a boost to their VRF score. This bid is burnt if they are selected as leader for the slot, whether the block is finalized or not. We expect this bid amount to reflect the amount of MEV extracted from public txs in the block, so burning it ensures the value is not entirely captured by builders, but rather shared across the network (insert ultra-sound meme here). Note that there's more economic analysis needed before deciding on this option.

Backup phase

To improve liveness, we propose adding a "backup phase" at the end of cycle if no block has been successfully proven, or if no prover commits to proving the block. During this phase, we go full based rollup, where anyone can submit an L2 block to L1. This should discourage sequencers who control two slots in a row from withholding the first block to profit from better MEV opportunities (we assume that more time and more txs equals to more value to extract), since a competitor would be able to capture the MEV from the first batch during this phase.

Merging phases

Still to be defined, but it'd be good to minimize the number of L1 txs we need to send per L2 block. We could possibly merge some of the phases as they are currently outlined, at least in the happy path.

Open problems

What we are still figuring out.

Proof withholding as an option purchase

Since individual tx proofs (think "tx signatures") are not posted to L1 to save gas, a sequencer may post a tx for which its proof is not available in the p2p network. This can be caused by a tx coming from a private tx pool, or done on purpose by the sequencer as a means to "purchase an option".

Assuming a vertically-integrated sequencer and prover, the sequencer can include a tx with a massive trade, and it retains full control during the entire slot (~5-10 minutes) on whether to actually "commit" that trade (by proving the block) or "roll it back" (by never disclosing the tx proof). This gives the sequencer an "option" over the asset being traded.

The cost of purchasing the option is equal to the burned sequencer bid, plus the prover commitment, plus potentially a fixed slashed stake. While there is a cost to not exercising it (the missed rewards, though the sequencer can still submit the block without

that trade during the backup phase to reap those), the earnings could well outweigh that cost.

Note that this problem exists in Ethereum as well, since a proposer can choose to reveal their block or not, but in Ethereum's case that window lasts 4 seconds. Also note that competition caused by the sequencer bid would not fix this problem either, since after the "option" has been purchased, there is no incentive towards committing the block.

A solution here is adding an L2 consensus or a data availability layer that verifies or at least guarantees the availability of the individual tx proofs, but this goes strongly against the goal of simplifying the protocol.

L1 congestion slashing honest sequencers

We know we want to punish a sequencer if their block doesn't get finalized by the end of their cycle, in order to promote liveness. This can be either through a fixed slashing amount, or by virtue of them losing their bid amount. However, an honest sequencer can fail to submit the block contents or proof to L1 due to network congestion, and get penalized anyway. We can mitigate this with longer windows, but this increases block times in the scenarios where sequencers withhold their block.

Edit 1: Updated to reflect @cooper-aztecLabs's comments below