Title

Cookie Jar

Proposed by

Zac Williamson, Cooper Kunz

Summary

We propose a privacy-preserving sequencer selection algorithm that employs a bidding mechanism to determine the right to create a rollup in a fully decentralized network, based off of Ethereum 2.0's current proof of stake mechanism. In Aztec, block producers must be known ahead of time as they coordinate block production (i.e., they post block state to the L2 gossip network, accept proofs from the proof pool, and coordinate rollup production). This proposal prioritizes the following characteristics: 1) ensuring the cost of block censorship is proportional to the block's economic value, 2) maintaining a predefined sequencer selection, and 3) preserving the privacy of sequencer identities. Also perhaps notably the proposal suggests the protocol can be built on both public and private Noir contracts.

Contents

Details

This proposal can be broken down into the following steps -

1. Registration
2. Bidding for block production rights
3. Sequencer selection
4. Proving block production rights

Registration

Sequencers are required to register against a specific public/private key pair at least one epoch in advance of making any bids. The verifiable random function's (VRF) salt (vrf_salt

) is updated every time a sequencer bid is made, rendering its value unknown at the time of registration. This precaution prevents a sequencer from testing random private keys until they discover one that yields a favorable VRF output. There is a question of how to bootstrap, or determine the initial sequencer. There are a few options here, including hardcoding a list of "initial nodes", coordinating a bootstrapping or randomized ceremony, or perhaps just trusting Vitalik to run a "bootnode" that is the first sequencer (this is a joke, folks… unless?

)

Bidding for block production rights

A sequencer places a bid, denoted as bid_value

, in tokens for the right to create a block. If the bid_value

is positive, it will be burned if the sequencer is selected to produce a block, regardless of whether they choose to or fail to produce a block for any reason. If the bid_value

is negative, the selected sequencer will only receive their tokens after successfully producing a block. A negative bid_value

exists to incentivize block production during periods of low use or inactivity such that the sequencer's costs exceed the revenue acquired from block fees. The maximum negative size of bid_value

is capped to prevent the case where, if the network only has a single sequencer bidding, they can create arbitrary inflation.

Sequencer selection

The selection algorithm, implemented as Noir contracts, encompasses both private and public transactions. It employs a private Noir contract and a VRF to generate a private, shielded_id

, which is subsequently added to a public Noir contract alongside the bid. This contract then sorts the shielded_ids

and bids, prioritizing the largest bids.

Simultaneously, the VRF output, while uniquely identifying a bid, also alters it to create random-weighted bids. This addition of randomness into the bidding process hampers monopolization of block production by ensuring that the highest bid only increases the probability, not the certainty, of selection for a given block.

An epoch is divided into slots, or fixed periods of time during which a sequencer can produce blocks. The epoch's bids are sorted by the value of their random-weighted bids, with the largest ones taking precedence and the largest negative ones going last. Each slot is then allocated to a sequencer based on these sorted bids. For a more detailed explanation, please refer to the algorithm section at the end of the proposal.

Proving block production rights

The selected sequencer must coordinate with provers to generate a block using an L2 messaging network. To ensure the provers recognize the correct sequencer, the sequencer will generate a temporary public/private key pair for use while producing a block and create a zk-SNARK that verifies their status as the sequencer with the current slot's shielded_id

.

Algorithm breakdown & more details

Blocks are organized into epochs. The expected block production rate is defined as block_rate

(in blocks/s). An epoch represents a period of epoch_length

seconds, sliced into epoch_length

- block_rate

slots. Each slot has a designated sequencer. At the end of epoch i, the sequencer list for epoch i + 2 is finalized, giving sequencers 1 epoch to 'get ready' to produce blocks.

N.B. all arithmetic in this section is integer arithmetic and not finite field arithmetic. Fractions are represented via both a numerator and a denominator to prevent a loss of precision.

Sequencer candidates use a VRF to pick a random number that determines their eligibility. The VRF consumes the following state:

- sequencer secret key sk
- epoch_number
- vrf_salt

The vrf salt is derived from all previous vrf values (defined below). The vrf_salt

value is updated every time a sequencer bid is made, making its value unknown at the time of registration. A sequencer bids a bid_value

amount of tokens for the right to create a block, which can be positive or negative, but not exceed a max_negative_bid

.

Private Noir Contract -

Initial VRF output is computed as:

VRF_{initial} = H(epoch_number + 2, sk)

Outputs of the private contract:

- VRF_{initial}
- bid_value

Public Noir Contract -

Let m = {block_rate} * {epoch_length}

(number of blocks in the epoch)

The contract tracks a sized-m array of slot

entries. Each slot entry is a tuple of { bid, shielded_id }

.

The array is sorted by bids (lowest first). Positive bids take precedence over negative bids (i.e. flip the sign!)

The contract computes m

VRF outputs:

VRF_{i} \text{ } \text{ for } i \in [0, m - 1]

Where

VRF_i = H(VRF_{initial}, salt, i)

This is used to produce m

bids where

bid_i = VRF_i * \frac{total_supply}{bid_value}

The bids are used to produce m

new slot entries, where shielded_id = VRF_i

.

A new slot size-2m slot array is computed and sorted. The first m entries are used to define the new slots

for the target epoch.

Updating salt

The public contract updates its salt value to be:

salt = H(salt, VRF_{initial})

Proving you're a valid sequencer

The chosen sequencer must coordinate with provers to generate a block using an L2 messaging network. A sequencer will perform the following steps:

1. Generate a temporary public/private key pair pk_{slot}, sk_{slot}

to identify themselves while producing a block.

1. Produce a ZK-SNARK proof that establishes:
2. They know the pre-image of the slot's shielded_id
3. They can sign over the pre-image with sk_{slot}
4. The outputs of the proof are { pk_{slot}, shielded_id }

Committing to a block

It is important that a sequencer commits to the transaction ordering within their block prior to requesting proofs be generated

from the Prover network (to prevent wasted Prover computation).

The sequencer must commit to a block hash on the L1 rollup contract, that is linked against pk_slot

and the block number. Once committed to this cannot be changed.

When the eventual block proof is submitted on-chain, the block hash will be defined as a public input to the rollup circuit. This will be checked against the committed value generated by the block msg.sender

. If they do not match the block is rejected.

Sequencer Verification

When a sequencer publishes a block, they must choose a public key pk_coinbase

to identify the coinbase address.

The rollup circuit validates that the block contains one private call to the sequencer verification L2 contract.

Verification Private Contract Logic

1. For a given private input sk

, compute sequencer_id = H(epoch_number, sk)

1. Validate a signature signed by sk

over a message that contains { sequencer_id, coinbase }

1. Call the public verification contract with input params sequencer_id, coinbase

Verification Public Contract Logic

The current block number is used to compute the current epoch's slot

.

From this, the slot's sequencer_id

is extracted.

The contract validates that the input sequencer id matches this value.

The contract validates that the value of coinbase

provided to the rollup circuit matches the input parameter coinbase

value.

Epoch Cleanup Logic

Cleanup work is needed for epochs less than the current_epoch

. For old epochs, ensure positive sequencer bids are burned, and negative sequencer bids are credited to sequencer accounts if a valid block has been produced.

Initializing the Sequencer List per Epoch

It is crucial to have a valid sequencer list for a given epoch. Although rare, it is necessary to rule out. One solution is to initialize the sequencers for epoch i + 2

as the same sequencers as i, but linked to a sequencer_value that is maximal, so any sequencer bids will overwrite the initial list.

L1 Contract Logic

The L1 contract uses its timestamp value to validate that the block's epoch number and slot number are correct.

Comparisons

This proposal strikes a balance between Joe's PBS design, which leverages a market-based solution to decentralize concerns, minimizing the need for a distinct "sequencer," and Cooper's Whisky proposal, which employs a random shuffle mechanism without an economic bid/burn marketplace. However, it does choose to leverage the L2 itself, and Noir, which is quite different! Alternative proposals, such as shared sequencers and L1-based coordination, pose challenges regarding MEV extraction and cost-efficiency.

As highlighted in the summary, this proposal prioritizes: 1) making block censorship cost proportional to the block's economic value, 2) upholding a predefined sequencer selection for proving efficiency, and 3) protecting the privacy of sequencer identities, presenting an effective balance among the options proposed thus far.

Drawbacks

This sequencer selection proposal suffers from an issue also present in Eth 2.0. If a sequencer is assigned to produce the final block in an epoch, whether they produce a block or not will affect the value of the VRF salt for the next bidding cycle.

Thus, if the sequencer bids for the next epoch, they can choose the VRF salt value that maximizes their chance of success not producing their block.

This issue is exacerbated if the final x

sequencers in a block choose to collude to maximize their chances of being selected in the next block.

This method of choosing the sequencer depends on a relatively large number of competing sequencers, such that the expected return from not producing a block is less than the value lost by not producing a block.

Feasibility

Given that a significant portion of this proposal can be implemented in Noir contracts, making it a pure engineering exercise (rather than a cryptography one) this will speed development. Given the complexity I think 6 months is an optimistic but fair estimate.

Glossary

- Epoch: In the context of blockchain, an epoch refers to a fixed time period during which certain actions take place. In this proposal, epochs are periods during which sequencers can produce blocks.
- Block: A block is a collection of transactions bundled together. In the blockchain, blocks are chained together to form the full transaction history of the network.
- Block Rate: This is the rate at which new blocks are added to the blockchain. It can be defined as blocks per second (blocks/s).
- Slots: In this proposal, slots are fixed periods of time within an epoch. Each slot has a designated sequencer who can produce blocks.
- VRFs (Verifiable Random Functions): VRFs are cryptographic functions that generate a random output that can be verified with a public key. In this proposal, they are used to add randomness to the bidding process for block production rights, making it difficult for a single entity to monopolize block production.
- VRF Salt: In the context of this proposal, the VRF salt is a value that is updated every time a sequencer bid is made. It is used as input to the VRF to generate a unique random number for each bid. The value of the VRF salt is unknown at the time of sequencer registration, making it impossible for a sequencer to manipulate the bidding process by finding a private key that yields a favorable VRF output.

Questions

1. Should there be max_negative_bid

? If so, what should it be?

1.a. Yes, the bid value should not exceed a max_negative_bid

(tbd) to prevent: Long-tail bids from excessively inflating the token supply, and in the event an attacker launches a targeted DDOS attack against all online sequencers and submits the only valid bid, protection against that negative bid being large enough to justify the attack.