Abstract

We propose a decentralized ZK-Rollup design based on the blob transactions described in EIP-4844 [1]. Each blob transaction contains a list of L2 transactions and a ZK proof to prove their validity. L2 builders compete to submit such blob transactions to L1 in a permissionless manner. When a blob transaction is successfully included in an L1 block, builders download the blob to update the L2 state.

Background

Decentralized ZK-Rollups

Current solutions to ZK-Rollups are mostly centralized: only specific operators can submit transaction batches and validity proofs to L1. Vitalik [2] gives several suggestions to make the operator decentralized, like sequencer auction, random selection from PoS set, and DPoS voting.

Polygon Hermez proposes a decentralized ZK-Rollup solution named PoE (Proof of Efficiency) [3]. In PoE, batch and proof submissions are permissionless and asynchronous. That means invalid batches may be submitted, which later have to be skipped or punished, making the proof submission process complicated.

StarkNet proposes a solution [4] to introduce the Tendermint consensus to L2, selecting the operator from a PoS set. But L1 should trust signatures from the operator set, making the solution trusted and thus not fully decentralized.

EIP-4844

EIP-4844 introduces a new transaction type, called “blob transactions”. Each blob transaction carries a “blob”, which is a user-defined piece of data whose commitment can be accessed by EVM. Blobs are downloaded by all beacon nodes and deleted after a relatively short delay.

For ZK-Rollups, storing batches in the blob is much cheaper than calldata. But how to access all historical batches to recover the whole L2 state in the “exodus mode” becomes a problem. If simply relay on a committee to store historical data like Validium, the solution becomes trusted and it provides less data availability than the calldata solution.

PoVP (Proof-of-Validity-Proof)

We design a decentralized ZK-Rollup solution based on EIP-4844, named PoVP. This solution is trustless and permissionless, which means anyone can submit L2 batches and proofs to L1.

In PoVP, a blob transaction contains both a transaction batch and a ZK proof to prove the validity of the batch. The blob transaction calls the rollup contract on L1 and updates the state root in it. Any rollup operators (called L2 builders in our solution) can build such a blob transaction and submit it to L1.

[

image

838×323 14.6 KB

](https://ethresear.ch/uploads/default/original/2X/5/5dbe414a994e92b15c65ba1bc3397a4e28e2b9fe.png)

Each L2 builder maintains:

1. All historical batches and the current state
2. A transaction mempool with a P2P network connected to other builders and users
3. A certain amount of computing power for generating ZK proofs

Upload batches from L2 to L1

It takes the following steps for an L2 builder to build a blob transaction:

1. Obtain a list of transactions from the mempool and form a batch
2. Run the batch on the current state to get the final state
3. Generate a ZK proof for the state transition

4. Generate a KZG commitment for the batch
5. Build a blob transaction with the above data and specified gas

Then the blob transaction is submitted to L1. If there are multiple blob transactions with the same pre-state root being submitted, the first one included in the blockchain will successfully update the state root in the rollup contract and others will fail. Therefore, L2 builders compete for the next batch and can take several strategies to win:

1. Accelerate the ZK proof generation by increasing its hardware or improving its software
2. Reduce the number of transactions in the batch, leading to faster ZK proof generation but fewer L2 gas rewards paid back
3. Pay more L1 gas to the blob transaction to make it included in the blockchain earlier

It's a tradeoff between costs (hardware, software, L1 gas) and rewards (L2 gas or other rewards for winning a batch). Thus each builder needs to develop its own optimal strategy. On the other hand, we can find that winning the next batch is mainly determined by the computing power, so builders tend to set a "mining pool" to aggregate power from a large number of hardware devices like GPUs and FPGAs, as well as develop a parallel/distributed architecture for computing ZK proofs. In addition, as the computing power raises, the L2 TPS also increases since more L2 transactions can be proven during a certain time period.

Download batches from L1 to L2

If an L2 builder detects a state update on L1, it should download the corresponding blob transaction and update its local state of L2. This ensures the data availability of the historical data in a trustless manner. You don't need to trust any L2 builders to store the whole state, because builders cannot build a new batch without it. It's similar to the data availability in a PoW blockchain where miners will voluntarily store the blockchain data for mining.

Advantages & Disadvantages

Advantages:

1. Fully decentralized: anyone can be an L2 builder, as long as they maintain the L2 state and have the computing power to generate the ZK proof
2. Gas saving: store batches in the blob transactions whose gas costs are much lower than calldata.

Disadvantages and possible solutions:

1. If multiple blob transactions with the same pre-state root are submitted to L1, the failed ones will still be included in the blockchain and thus waste storage space. This can only be solved if the L1 block builders can reject failed blob transactions.
2. The TPS of L2 is relatively low because transaction batches are proven serially. Maybe it is possible to prove transactions within a batch parallelly to increase the TPS.
3. Only one L2 builder will win the next batch, so the computing power from other builders is wasted. A ZK proof mining pool is needed to aggregate the computing power.

References

[1] EIP-4844: Shard Blob Transactions

[2] An Incomplete Guide to Rollups

[3] Proof of Efficiency: A new consensus mechanism for zk-rollups

[4] StarkNet Decentralization - Tendermint based suggestion - StarkNet Shamans