Opside ZK-PoW Introduction

Opside is a decentralized ZK-RaaS (ZK-Rollup as a Service) platform and a leading network for ZKP (Zero-Knowledge Proof) mining. ZK-RaaS provides a one-click service for generating ZK-Rollups to anyone. Opside offers a universal ZK-Rollup launchbase, allowing developers to easily deploy different types of ZK-Rollups on various base chains.

- Base chains: Ethereum/Opside chain/BNB chain/Polygon PoS, and other public chains.
- ZK-Rollup types: zkSync, Polygon zkEVM, Scroll, StarkNet, and other variants of ZK-Rollups.

[

1280×700 238 KB

](https://ethresear.ch/uploads/default/original/2X/7/77cd85e53f4917788079fb31a9d1922a9653d000.png)

In Opside's design, developers can deploy ZK-Rollups on these different base chains. As ZK-Rollup technology matures, there may be hundreds or even thousands of ZK-Rollups in the future, which will create a significant demand for ZKP computation power. Opside utilizes the ZK-PoW mechanism to incentivize miners to provide ZKP computation power, thus providing a complete hardware infrastructure for ZK-Rollups.

ZK-PoW V2.0 Architecture

The overall architecture of ZK-PoW V2.0 consists of several key components:

1. ZK-PoW Cloud: This is the cloud infrastructure provided by Opside for ZKP computation. It is deployed across multiple chains, including Ethereum, BNB Chain, Polygon PoS, and Opside Chain. The ZK-PoW Cloud is responsible for coordinating and managing the ZKP computation tasks.
2. Miner Nodes: These are the nodes operated by miners who contribute their computational power to perform ZKP computations. Miners can participate in the ZK-PoW network by running specialized software on their mining hardware.
3. ZKP Task Distribution: The ZK-PoW Cloud distributes ZKP computation tasks to the miner nodes. The distribution is done in a decentralized manner to ensure fairness and efficiency. The ZKP tasks include generating and verifying zero-knowledge proofs for various ZK-Rollups.
4. ZKP Computation: The miner nodes receive ZKP computation tasks and perform the necessary computations to generate the required proofs. This involves executing cryptographic algorithms and performing complex calculations.
5. Proof Submission and Verification: Once the ZKP computations are completed, the miner nodes submit the generated proofs to the ZK-PoW Cloud for verification. The cloud infrastructure verifies the correctness of the proofs to ensure their validity and integrity.
6. Incentive Mechanism: Miners are incentivized to participate in the ZK-PoW network by earning rewards for their computational contributions. The reward system is designed to motivate miners and maintain the security and stability of the network.

Overall, ZK-PoW V2.0 combines the power of miners' computational resources with cloud infrastructure to provide efficient and scalable ZKP computation for a wide range of ZK-Rollups.

The Aggregator is an important component of the Prover in the ZK-PoW V2.0 system. It is responsible for distributing ZKP proof tasks, receiving task results (ZKP proofs), managing the proofs, and submitting them to the Base Chain to earn rewards. Based on their functions, the new version of the Aggregator is divided into three sub-modules: Proof Generator, Proof Manager, and Proof Sender.

[

1440×1440 174 KB

](https://ethresear.ch/uploads/default/original/2X/3/36b20ae761b85ed965ce65b9e478b9de1b621697.jpeg)

- The Proof Generator module is responsible for assigning proof tasks to the Prover (PoW miner), receiving the task results (ZKP proofs), and storing the ZKP proofs in the DB database.
- The Proof Manager is in charge of managing the completed ZKP proofs and packaging the proofs that are ready for on-chain submission as tasks for the Proof Sender module.
- The Proof Sender module handles the on-chain submission of ZKP proofs by submitting them to the zkevm contract deployed on the Base Chain.

Below are the introductions of these three modules:

Proof generator

Rollup Chain aggregates a certain number of transactions into a batch, and then multiple batches (based on factors such as transaction frequency) are combined into a sequence. The sequence is then submitted to the Base Chain, making it the unit of data submission for each on-chain operation. Each sequence consists of one or more batches, and the ZKP proof verifies the validity of the submitted sequence. Therefore, the batch is the smallest unit of proof task.

Depending on the number of batches included in a sequence, the required proof tasks vary as follows:

- If the number of batches is 1, the proof process involves BatchProofTask followed by FinalProofTask, and the tasks are completed sequentially.
- If the sequence contains more than 1 batch, the proof process involves multiple BatchProofTasks, an AggregatorProofTask, and a FinalProofTask, and the tasks are completed sequentially.

To maximize the efficiency of proof generation and increase the mining rewards for PoW miners, we aim to generate proofs concurrently. This is achieved in two aspects:

- Proof generation for different sequences can be done concurrently as there is no contextual or state dependency.
- Within the same sequence, multiple BatchProofTasks can be executed concurrently.

This approach utilizes the computational resources of Provers more efficiently, resulting in more efficient proof generation.

Proof manager

This module is primarily responsible for managing ZKP proofs and controlling their on-chain verification. It consists of three main modules:

- submitPendingProof: This module is executed only once when the Aggregator starts. Its purpose is to complete the submission of unfinished ZKP proofs from the previous Aggregator service. This handles the situation where proofHash has been submitted but other miners have already submitted their proofs. For more information about proofHash, please refer to the Proof Sender module.
- tryFetchProofToSend: This module runs as a coroutine and adds the latest generated ZKP proof, along with its corresponding unverified sequence, to the Proof Sender’s cache, waiting for on-chain submission.
- processResend: This module runs as a coroutine and aims to resubmit sequences that have not been successfully verified within a given time window. Its purpose is to ensure that sequences that exceed the verification time are resubmitted for on-chain processing.

Proof sender

Opside has proposed a two-step ZKP submission algorithm to achieve decentralization of the prover. This algorithm prevents ZKP front-running attacks and enables more miners to receive rewards, thereby encouraging more miners to participate and provide stable and continuous ZKP computation power.

- Step 1: When producing a PoW proof for a specific sequence (referred to as proof), the prover first calculates the hash of “proof / address” and submits it to the contract. If no proofHash has been submitted for that sequence before, a proofHash submission time window, T1, is opened. Miners are eligible to submit the sequence within T1 blocks, and proof can only be submitted after T1 blocks.
- Step 2: After T1 blocks, the proof submission window is opened for T2 blocks. If none of the submitted proofs pass the verification within T2 blocks, all miners who previously submitted proofHash will be penalized. If proofHash is successfully submitted within the T1 time window but proof cannot be successfully submitted within the T2 time window, and other miners successfully submit their proofs within T2 window, the prover can still continue to submit that proof. In other scenarios, the two-step submission process is restarted.

The following diagram illustrates how Proof Sender implements the two-step submission using three thread-safe and priority-sorted caches. These caches are sorted based on the starting height of the sequences, ensuring that each time an element is retrieved from these caches, it corresponds to the lowest sequence height. Additionally, the elements in these caches are deduplicated. The lower the height of the corresponding sequence, the higher the priority for processing.

- finalProofMsgCache: Stores the finalProof messages sent by the Proof Manager, indicating the completion of ZKP proofs.
- monitPHTxCache: Stores the proofHash transactions to be monitored.
- ProofHashCache: Stores the proof messages for on-chain submission.

[

1440×1440 117 KB

](https://ethresear.ch/uploads/default/original/2X/5/568544c70193e48fe6c090ec5d4d94e5d2e7b186.jpeg)

After the Proof Sender module is launched, three coroutines are started to consume data from the three caches. The simplified process is as follows:

1. Coroutine 1 consumes the finalProof messages from the finalProofMsgCache, calculates the proofHash, and if it meets the conditions for on-chain submission (within the T1 window), it submits the proofHash to the chain and adds the proofHash transaction to the monitPHTxCache.
2. Coroutine 2 consumes the proofHash transaction messages from the monitPHTxCache. If the proofHash meets the conditions for on-chain submission within the T2 window, it constructs the proof message and stores it in the ProofHashCache.
3. Coroutine 3 consumes the proof messages from the ProofHashCache and submits the proofs to the chain.

Compared to the previous module, this structure is clearer and saves on resource overhead.

Summary

Comparison with Version 1.0

- Version 2.0 splits the original service into three sub-modules, each responsible for proof generation, proof management, and proof submission, resulting in a clearer structure, lower coupling, and stronger robustness.
- The proof generation module, Proof Generator, has added the startBatch parameter compared to the old version, making it easier for new miners to catch up with the mining progress.
- The proof management module, Proof Manager, has been improved compared to the old version. It promptly resends proofs in case of miner service restart or other reasons for proof submission failure, ensuring miner interests. The resend mechanism not only addresses proof submission failure cases but also handles all cases of proof submission failure or non-submission, ensuring the security of the Rollup Chain.
- The proof submission module, Proof Sender, implements a two-step transaction submission using three thread-safe priority caches. It reduces the use of global locks compared to previous versions, ensuring that proofs with lower heights are submitted promptly and protecting miner interests. Additionally, the overall service flow is clearer, with reduced thread count and reduced resource consumption during program execution.

Stress testing results:

- In Version 2.0, using 10 machines with 64 cores each, 566 batch proofs were completed in 7 hours, 38 minutes, and 40 seconds, with an average time of 48.62 seconds per proof. In a multi-miner scenario, compared to Version 1.0, the efficiency of zk proof generation in Version 2.0 improved by 50% overall.

In conclusion, Opside ZK-PoW V2.0 has optimized the process of multiple miners participating in ZKP computations, improving hardware utilization, enhancing service availability, and being more miner-friendly. Importantly, in a multi-miner scenario, the computation time for ZKP has been reduced to less than a minute, significantly accelerating the confirmation time of ZK-Rollup transactions.