ZKP BASED LIGHT CLIENT RESEARCH

By Zpoken team

Monthly progress (May)

Zpoken is building the Near ZK Light Client, intended to address both on-chain and off-chain applications. The current blockchain architecture has unquestionably benefited from trustless, or non-custodial, asset ownership. However, users must rely on the public state provided by a centralized and unverified counterparty (RPCs). This reliance is due to the prohibitive cost and complexity of running one's own node in the case of off-chain operations, and the excessive costs and limitations of existing chains for on-chain applications.

To tackle these issues, light clients were introduced, but they still demand substantial computational resources and syncing time. These requirements make them challenging for off-chain use and almost unfeasible for on-chain scenarios. This is where zk light clients come into play, offering provable state and aiding both off-chain and on-chain applications to leverage interoperability and trustless data exchange.

A significant advantage of the zk light client over the traditional light client is the reduction in the size of data and computational resources needed for verification. With classical light client, we must start from genesis (or the block height where we last stopped) and compute headers sequentially up to the target, possibly spanning millions of blocks. In contrast, a zk light client can verify the chain from genesis (or the previously verified block) up to the target with just one proof and a few public input hashes. This results in considerably lower verification complexity compared to a conventional client, ultimately reducing the overall computational demands within the ecosystem.

The Near ZK Light Client is designed to achieve these goals and foster a more decentralized and trustless ecosystem by proving epoch and regular blocks. With a verified block hash, the light client can confirm different public states by verifying the Merkle proofs included in the verified block hash.

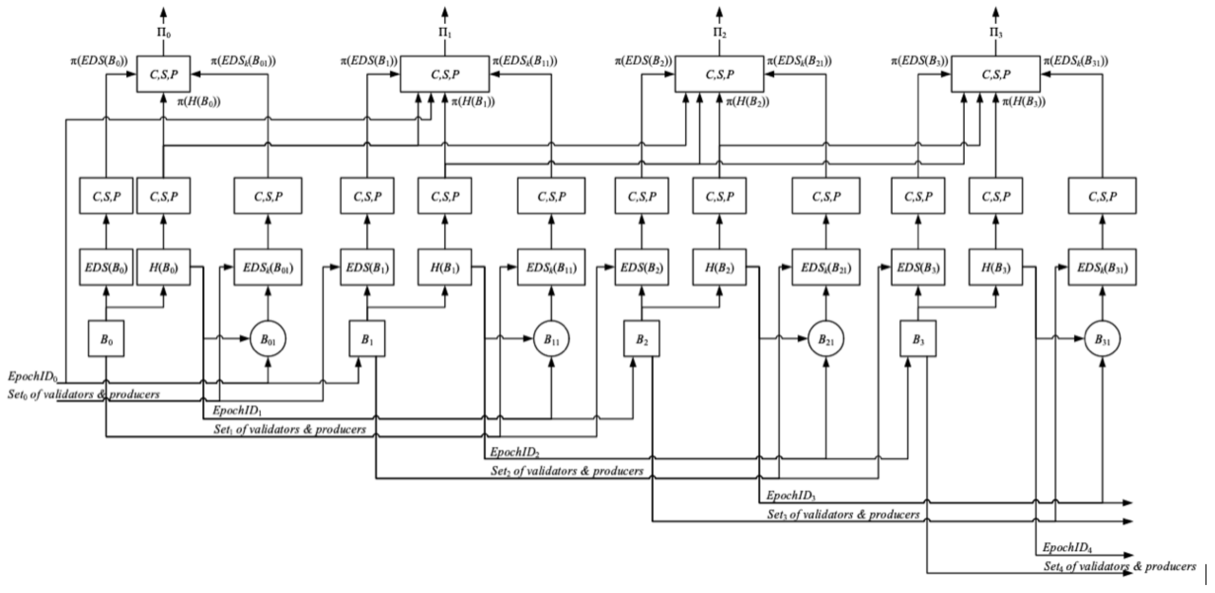
The Near Protocol features epochs, each with a preset list of 100 validators and producers authorized to produce, validate, and finalize the regular blocks. Each epoch consists of 43,200 regular blocks, and every block commits the hash of the validator list used in the N+2 epoch. Therefore, to verify whether a block in epoch N has been signed with the correct validator list, we must refer to the blocks from the N-2 epoch and verify the committed next_bp_hash (assuming the finality of this block in the Nth epoch has already been checked).

When we discuss proving an epoch block, we mean computing a block hash with block header data that includes the previous block hash, next_bp_hash, and other relevant information about state transitions made in that block. We also compute signatures for this block and verify their public key commitments in the N-2 epoch. These operations are broken down and proven in our Plonk2 circuits using cryptographic gadgets like SHA-256, SHA-515, and Ed25519.

Our current implementation of the Near ZK Light Client confirms the correctness of block hash computations. We prove a block or sequence of blocks for correct hash computation, validate a validator list committed in the epoch block N-2, and confirm the signature.

Presently, we are incorporating all signature proving into the block and verifying their inclusion in next_bp_hash. Once this goal is achieved, our next step is to update this configuration to prove regular blocks based on the same epoch block proving methodology and move to on-chain verification to power web3 applications with verifiable data.

1. Proving of epochal blocks
    1.1. Developed a scheme for next_bp_hash and approval signatures.



    1.2. Implemented bp_hash proof
    1.3. Tested implementation on 100 epoch blocks and own test vector.
    1.4. For real block testing, parsed NEAR blockchain via JS requests and retrieved blocks, signatures, approvals.
    1.5. After benchmarking, chose chunk size in range 5-8 for optimal parallel execution of map-reduce.
    1.6. Implemented new sha256 circuit with efficient memory usage with parallel operations in the field.
2. Results
    2.1. Benchmarking for bp_hash proving
    5 blocks - 99.8s
    100 blocks - 2004.3s
    2.2. Implementation