Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

ZKP BASED LIGHT CLIENT RESEARCH

By Zpoken team

Monthly progress

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

DOCUMENT GUIDE

SECTION I: Research on epochal blocks

We have conducted research on epochal blocks in the NEAR blockchain and developed a script for parsing them.

The parsing script is available on GitHub.

The parsing results (100 consecutive epoch blocks, the first and last blocks of each epoch) can be found at:

https://github.com/Valentine1898/near-block-parser/tree/master/blocks

The block names are formatted as:

`block-${height}-${first\last}-${epoch_id}.json`

Research focused on the hash block of producers in the block (`next_bp_hash`).

The `next_bp_hash` value is stored in each epoch block. This is the hash of the next epoch block producers set. This hash is calculated using `hash_borsh_iter()` function:

```
pub fn hash_borsh_iter<I>(values: I) -> CryptoHash
  where
    I: IntoIterator,
    I::IntoIter: ExactSizeIterator,
    I::Item: BorshSerialize,
  {
    let iter = values.into_iter();
    let n = u32::try_from(iter.len()).unwrap();
    let mut hasher = sha2::Sha256::default();
    hasher.write_all(&n.to_le_bytes()).unwrap();
    let count = iter.inspect(|value| BorshSerialize::serialize(&value, &mut hasher).unwrap()).count();
     assert_eq!(n as usize, count);
```

The set ValidatorStake:

https://github.com/near/nearcore/blob/b056967b3c6646ffc5685a93f163bd869ec1caac/core/primitives/src/types.rs#L480 is passed as parameters `hash_borsh_iter()` function.

Worthy of mention: https://nomicon.io/ChainSpec/LightClient#signature-verification is the description of the block signature verification algorithm

SECTION II: Implementation of proving epochal blocks (hashing only)

The general scheme for the formation of a chain of recursive proofs is shown in fig. 1.

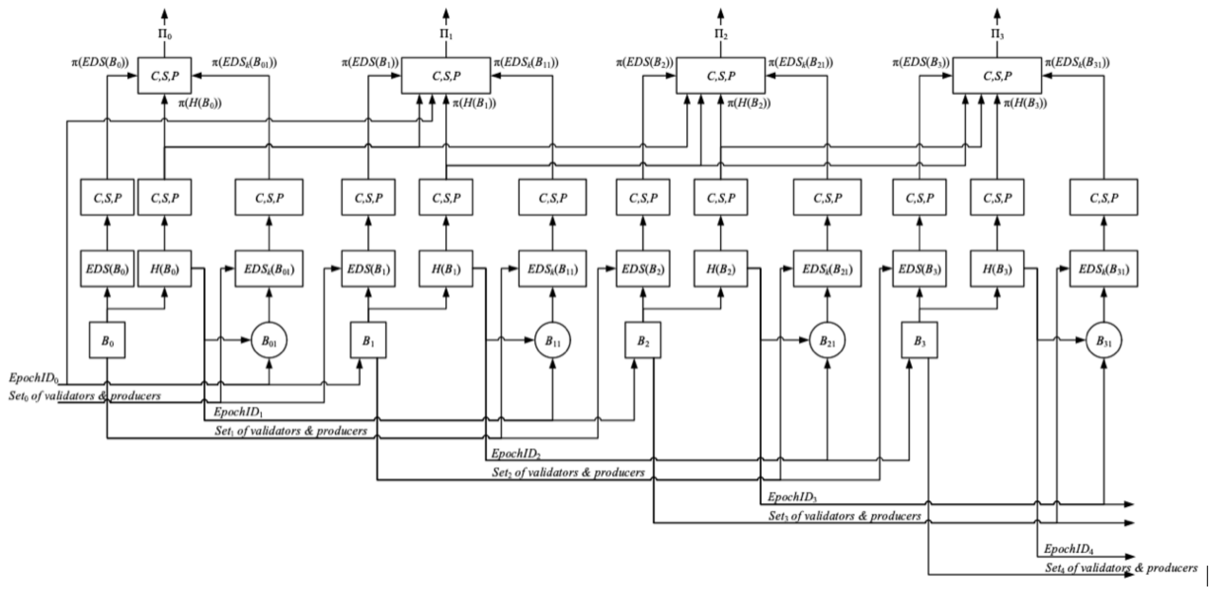

Fig. 1 - Cryptographic chain of epochal blocks (simplified scheme)

In the implemented case we use real block structures with proving hashes for epochal blocks, i.e. proof generation $\pi(H(B_i))$ of the computational integrity of a hash $H(B_i)$ of each epochal block:

Epochal block structure:

Implementation:

We have released to GitHub repository our implementation of circuits with caching optimizations that give us up to 2sec to prove a block (and there is still ways to optimize more), our benchmarks, test sample of 100 epoch blocks and corresponding scripts for blocks parsing from a node and processing them before proving.

https://github.com/ZpokenWeb3/zk-light-client-implementation

SECTION III: Research for plonky2 proof verification with circom

Optimizing the recursive epoch block proof scheme to minimize the complexity and cost of publishing a smart contract (take a slow/small plonky2 proof and invoke rapidsnark with it and its PIs, and we have followed the approach what PolymerDAO “plonky2-circom” does in its example for that).

Implementation:
We have used PolymerDAO “plonky2-circom” implementation to prove the correctness of sha256 hash calculation proof made with plonky2. However we faced compilation issues (circuit failed to compile, even if we roll back the plonky2 version). We had asked PolymerDAO team about this issue in their discord channel, but haven’t received reply yet.