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

ZKP BASED LIGHT CLIENT RESEARCH

By Zpoken team

Monthly progress (February)

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

DOCUMENT GUIDE

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

INTRODUCTION

Our society is constantly evolving. Digital information storage systems, which provide reliable, safe and consistent reproduction of recorded facts and issues, are also being improved. However, we are still far from the model of interaction, where everyone has access to such services which provides anonymity, freedom of action, lack of censorship etc. All these requirements are a natural continuation of people's needs for democracy of decisions, privacy and independence from others. This is the desire for a fair and honest model of interaction.

In recent years, we have seen rapid progress in the development of digital technologies that partly meet these complex and often conflicting requirements. Today we have learned how to build decentralized distributed ledger (DLT) that store information on independent and unaccountable nodes that interact in complete distrust with each other, and yet provide reliable, secure and historically consistent storage. These important qualities are provided by a complex system of distributed computing, advanced cryptography and reliable communication protocols of peer-to-peer networks. In addition, modern blockchain projects provide many valuable services: smart contracts, digital tokens, integration of game universes, the Internet of things, and much more.

The main problem of modern DLTs is scalability. It is when, in addition to security and decentralization of information services, it is necessary to provide a resource availability service for billions of users in different parts of the world, including for fixing events in game universes, the Internet of things, the implementation of smart contracts, electronic voting etc. The load on computing resources and telecommunications components increases with the simultaneous access of a huge number of users and impersonal Internet queries of search bots. In addition, everyone who interacts with the blockchain network project in the mode of complete distrust forces to repeatedly check long chains of blocks containing cryptographic integrity and involvement marks of the entities that generated them. And this is the main restrictive reason for the further development of DLT. The more issues we capture in DLT, the longer the generated block chain is and the more difficult it is to check the integrity and security of the storage. Extensive development in this case does not help, but, on the contrary, reduces the effectiveness of already deployed DLTs. We solve this main problem of modern distributed registries through the use of new cryptographic engineering technologies, known as zero-knowledge proof systems (ZK-SNARK).

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

ZK-SNARK is a cryptographic proof that allows one party to prove it possesses certain information without revealing that information.

It allows speeding up generating proofs and proceeding verification.

It is possible to make a proof approximately in $O(t \cdot log(t))$ steps for computations that require t steps. Verification of such proof requires $O(log^2(t))$ steps.

In fact, ZK-SNARK replaces the native registry check with two calculations:

- It is possible for a prover to replace a complex process of generating proof with just precomputing and storing a resulting proof;
- It is possible for a verifier to implement a fast and low-resource verification for a light client.

In this document, we outline the main ideas of our solution and justify the general architecture of fast, reliable and secure DLT verification systems using ZK-SNARK in relation to the interaction protocols of the NEAR blockchain project.

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

SECTION II: Test case #3. Generating a chain of proofs for hashes and signatures of block producers, listed in the epochal block

For test case #3, we implement a recursive proof of the computational integrity of a chain of linked hashes with digital signature validation of block producers (there have to be at least ⅔ shares of the total amount of the current epoch). This is an important element of the NEAR protocol, designed to decentralize block decisions. In test case #3, we use the implementation from test case #2 to model this solution.

Initial data. Each block contains three fields:

1. Unique block number: $Nonce_i$;
2. Hash of the previous block: h_{i-1};
3. Digital signature: $EDS(h_i, sk)$, generated by block producer with a secret key sk.

According to the description of the NEAR protocol, each block contains signatures of other producers $EDS_i^j(h_i, sk^j)$ with secret keys sk^j. Since h_i is calculated over the i-th block, which includes h_{i-1}, EDS_i^j signatures are considered as a confirmation of the authenticity of the connection of two blocks. All EDS_i^j are stored in the header of the $(i + 1)$-th block (it is not shown in the figure) and are used to confirm the $(i - 1)$-th block.

A list of all block producers (pk and all pk^j) is published in the first block of the epoch. We assume that the stakes of all producers are equal to one, i.e. we need at least 2/3 shares for coalition confirmation of the i-th block.

A simplified scheme of the linked list with digital signatures of all producers is shown in the fig. 10.

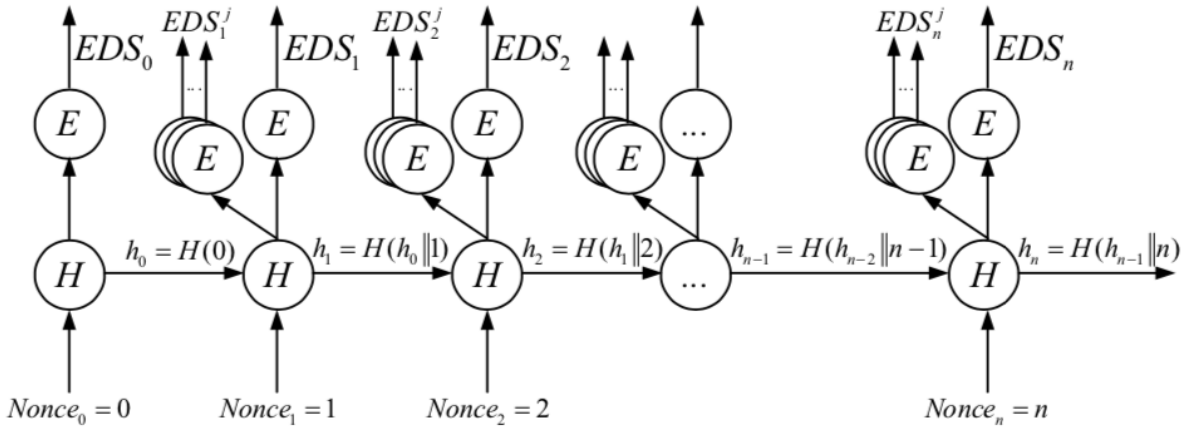

Fig. 1 – A simplified version of the linked list with digital signatures of all producers

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

SECTION III: Test case #3. Aggregation & Recursion methods (Recursion in progress)

Aggregation. A task is to implement a test case of a recursive proof of computational integrity of a chain of linked hashes with verification of the correctness of digital signatures of block producers. In other words, we need to prove the following for the *n*-th block :

1. CI of h_n;
2. CI of the digital signature verification $h_n = D(EDS_n, PK)$ of the current block producer;
3. CI of the digital signature verification $h_{n+1} = D(EDS_{n+1}^j, PK^j)$ of all producers from the epochal block list;
4. The share of verified signatures EDS_{n+1}^j is at least 2/3 of the total number of producers.

To simplify the task, we assume that there are three permanent block producers:

1. (SK, PK) – direct block producer;
2. (SK^1, PK^1) – the first member of the coalition (producer from the list of the epochal block);
3. (SK^2, PK^2) – the second member of the coalition (producer from the list of the epochal block).

Then for the *n*-th block we need to prove:

1. CI of h_n;
2. CI of $h_n = D(EDS_n, PK)$;
3. CI of $h_{n+1} = D(EDS_{n+1}^1, PK^1)$ and $h_{n+1} = D(EDS_{n+1}^2, PK^2)$;

The scheme of forming a chain of recursive proofs of CI with digital signature verification of all producers is shown in fig. 11. It presents the notation from example #2. To simplify it, we have grouped the stages of generating the computational scheme C, public parameters (S_p, S_v) and proofs and marked them as (C, S, P).

Each proof $\prod_i$ for all $i = 1,...(n - 1)$ is the result of aggregation of such proofs as:

1. proof of CI of the previous chain of linked hashes $\prod_{i-1}$;
2. proof of CI of the current hash πH_i;
3. proof of CI of digital signature verification of the current block producer πD_i;
4. two proofs of CI of digital signature verification πD_i^1 and πD_i^2 of producer from the list of epochal blocks.

Proof $\prod_0$ is the result of aggregation of the following proofs:

1. proof of CI of the current hash πH_0 ;
2. proof of CI of digital signature verification of the current block producer πD_0;
3. two proofs of CI of digital signature verification πD_1^1 and πD_1^2 of producer from the list of epochal blocks.

Thus, the circuit in fig. 11 simulates in a simplified scheme of the main idea of the NEAR protocol: each block is signed by the block producer, the block is confirmed by all other producers with their signatures of the next block. The scheme simulates a process for three producers (one who produces blocks and the other two confirm).

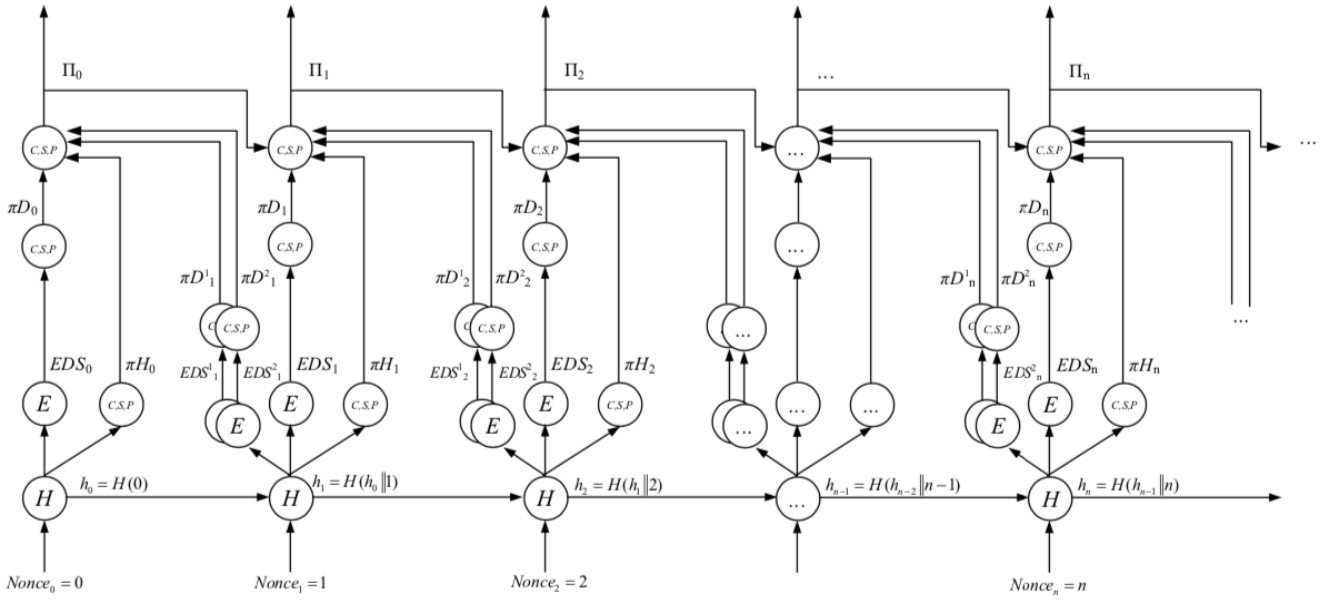

Fig. 2 – Scheme of formation of a chain of recursive proofs of CI with verification of the correctness of digital signatures of all producers

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

SECTION IV: Test case #3. Time and measurement results

All computations were made on 1,8 GHz Intel Core i5.

Aggregation. The scheme implementation:

https://github.com/tikhono/zkp-research/tree/proof_to_file/plonky2_test_case3

Table 1 — Time and measurement results for proofs for hashes (1,8 GHz Intel Core i5)

№	Hash	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9	2.1245308	4.3383	132640	0.0460
1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	4.4514556	7.9676	150268	0.0146
2	9C808663A3D1A47F3FCFE26BA11AFD90D0F00A41984F8058DE3CE8552C6BEF77	4.9265103	8.0290	150268	0.1134
3	B3CC82D30374FC4FA584032CF35E86A54E70E08A9E3E9982BFE06A6022A4937A	5.574602	8.3707	150268	0.0128
4	4D23AF03289C3EB09E4C888999102EE53E6A49D49FD06326A09ED924D22D34A9	4.8302417	7.7315	150268	0.0455

Table 2 — Time and measurement results for proofs for signatures (1,8 GHz Intel Core i5)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	83.94802	1308.5372	208376	0.0542
1	89.62305	2066.8973	208376	0.0171
2	90.434525	1723.5806	208376	0.0868

3	94.18398	2273.7472	208376	0.0177
4	91.60083	2005.1410	208376	0.0177

Table 3 — Time and measurement results for aggregated proofs for signatures & hashes for current blocks (1,8 GHz Intel Core i5)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	5.8441653	10.1804	146348	0.0296
1	4.2841697	8.1614	146348	0.0393
2	4.421095	8.1313	146348	0.0267
3	4.285184	8.2347	146348	0.0356
4	4.45687	8.3481	146348	0.0120

Table 4 — Time and measurement results for two other producers of signatures (1,8 GHz Intel Core i5)

№	Time to build a circuit *(pr1/prd2)*, s	Time to make a proof *(pr1/prd2)*, s	Proof size, bytes	Verification *(pr1/prd2)*, s
0	81.24202/81.63741	677.5956/1183.6942	208376	0.1520/0.0357
1	83.77755/85.18298	1598.8934/1948.9931	208376	0.0376/0.0166
2	103.31765/101.24621	1718.6047/1861.8560	208376	0.0884/0.0178
3	112.7302/120.990234	1841.7534/2084.3656	208376	0.0587/0.0169
4	–	–	–	–

Table 5 — Time and measurement results for a final aggregated proof chain (1,8 GHz Intel Core i5)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	15.397868	17.4154	152684	0.0329
1	11.60591	19.4106	152684	0.0336

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

2	17.610699	18.1004	152684	0.0517
3	12.636483	19.3275	152684	0.0468
4	8.105103	9.4573	146348	0.0134

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

Test case #4: recursive proof of the computational integrity (CI) of a blockchain with verification of the correctness of digital digital signatures of producers and validators.

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

1. Proof generation $\pi(H(B_i))$ of the CI of a hash $H(B_i)$ of each block;
2. Proof generation $\pi(EDS(B_i))$ of the CI of a digital signature $EDS(B_i)$ of the producer of each block. The list of block producers must match the list in *Set of validators & producers*;
3. Proof generation $\pi(EDS_k(B_i))$ of the CI of digital signatures of the validators of each block. The list of block validators must match the list in *Set of validators & producers*;
4. Proof aggregation:
    a. generating proof of CI $\prod_0 = (\pi(H(B_0)), \pi(EDS(B_0)), \pi(EDS_k(B_1)))$ of block by aggregating:
        i. a proof of the correct hashing $\pi(H(B_0))$ of the block B_0;
        ii. a proof of the correct digital signature $\pi(EDS(B_0))$ of the block producer B_0;
        iii. a proof of correct digital signatures $\pi(EDS_k(B_1))$ of block validators B_1;
    b. generating a proof of CI $\prod_i = (\pi(H(B_i)), \pi(H(B_{i-1})), \pi(H(B_0)), \pi(EDS(B_i)), \pi(EDS_k(B_{1+1})))$ of each block by aggregating:
        i. a proof of the correct hashing $\pi(H(B_i))$ of the block B_i;
        ii. a proof of the correct hashing $\pi(H(B_{i-1}))$ of the block B_{i-1};
        iii. a proof of correct hashing $\pi(H(B_0))$ of the block B_0, i.e. the correctness of the calculation of the epoch identifier $EpochId = H(B_0)$;
        iv. a proof of the correct digital signature $\pi(EDS(B_i))$ of the block producer B_i;
        v. a proof of correct digital signatures $\pi(EDS_k(B_{1+1}))$ of block validators B_{1+1}.

In the test case, we assume that the *Set of validators & producers* contains a list of 3 validators, each of which is the producer of the corresponding block, i.e. the first producer forms the block B_1; B_2 is formed by the second producer and the third one forms B_3.

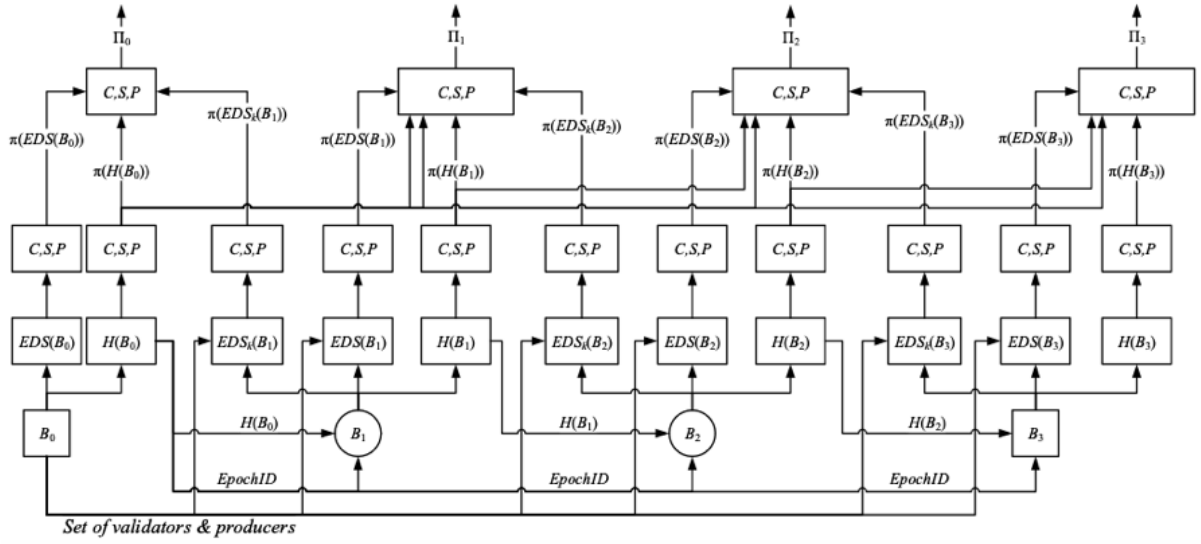

Fig. 1 – Cryptographic block chain (simplified)

The scheme in fig. 1 is a simplified model of the NEAR protocol. In particular, the most significant simplifications are:

- validation of each block requires 2/3 of the share (of the total stake) of the signatures of 100 validators of the epoch. To validate the last two blocks of an epoch, 2/3 of the share (of the total stake) of the signatures of 100 validators of this and the next epoch is required (Fig. 2 uses a fixed list of 3 validators);
- the epoch identifier is calculated as the hash code of the last block of the penultimate epoch (in Fig. 2, the hash code of the last block of the last epoch is used);
- to the private input of the proof generator $\pi(EDS(B_i)$ and $\pi(EDS_k(B_i))$ it is necessary to submit the *Set of validators & producer* from the last block of the penultimate epoch (this is not shown at all in Fig. 2).

The presented scheme can be further simplified by implementing only a check of the CI of epochal blocks (the remaining blocks of each epoch).

Implementation: https://github.com/tikhono/zkp-research/tree/test_cases/plonky2_test_case4

Test case #4.1: recursive proof of the computational integrity of the chain of epochal blocks with verification of the correctness of digital digital signatures of producers and validators

The general scheme for the formation of a chain of recursive proofs is shown in fig. 2.

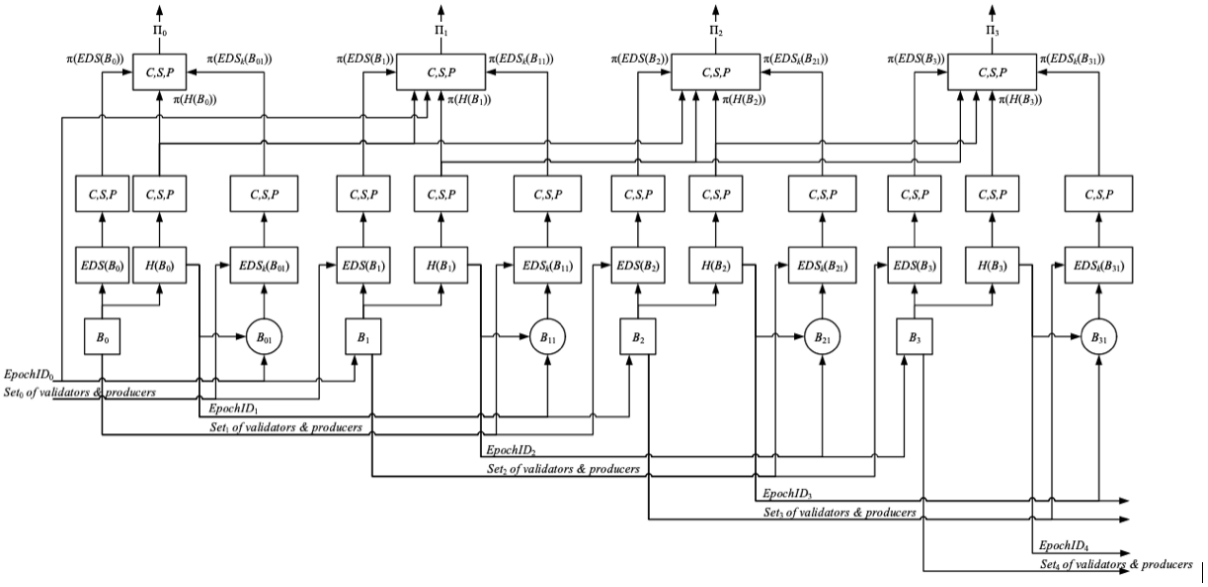

Fig. 2 - Cryptographic chain of epochal blocks (simplified scheme)

It includes:

1. Proof generation $\pi(H(B_i))$ of the CI of a hash $H(B_i)$ of each epochal block;
2. Proof generation $\pi(EDS(B_i))$ of the CI of a digital signature $EDS(B_i)$ of the producer of the epochal block. The list of block producers must match the list in *Set of validators & producers* of the penultimate epoch;
3. Proof generation $\pi(EDS_k(B_{i1}))$ of the CI of digital signatures of the validators of the first epoch block $EDS_k(B_{i1})$. The list of block validators must match the list in Set_{i-1} *of validators & producers* of the penultimate epoch;
4. Proof aggregation:
    a. generating proof of CI $\prod_0 = (\pi(H(B_0)), \pi(EDS(B_0)), \pi(EDS_k(B_{01})))$ of epoch block B_0 by aggregating:
        i. a proof of the correct hashing $\pi(H(B_0))$ of the epoch block B_0;
        ii. a proof of the correct digital signature $\pi(EDS(B_0))$ of the epoch block producer B_0;

iii. a proof of correct digital signatures $\pi(EDS_k(B_1))$ of first epoch block validators B_{01};

b. generating a proof of CI $\prod_i = (\pi(H(B_i)), EpochId, \pi(H(B_{i-1})), \pi(EDS(B_i)), \pi(EDS_k(B_{i1})))$ of each block by aggregating:

i. a proof of the correct hashing $\pi(H(B_i))$ of the epoch block B_i;

ii. $EpochId$ is an initial epoch identifier (constant);

iii. a proof of the correct hashing $\pi(H(B_{i-1}))$ of the epoch block B_{i-1}, i.e. the correctness of the calculation of the epoch identifier $EpochId_{i+1} = H(B_{i-1})$;

iv. a proof of the correct digital signature $\pi(EDS(B_i))$ of the block producer B_i;

v. a proof of correct digital signatures $\pi(EDS_k(B_{i1}))$ of block validators B_{i1}, i.e. first epoch block with id $EpochId_{i+1} = H(B_{i-1})$.

In this test case, we assume that the Set_{i-1} *of validators & producers* contains a list of 3 validators for the epoch with id $EpochId_i$. Each validator is the producer of the corresponding block as in test case 4.

Note, that:

- the epoch block is the last block of the epoch with id $EpochId_{i-1}$, it contains the list of Set_{i-1} *of validators & producers* of validators and producers for the epoch with id $EpochId_{i+1}$;
- for epochs with identifiers $EpochId_0$ and $EpochId_1$ the list of validators and producers is predefined (not specified in any blocks).

It should be noted that when forming the proof $\prod_i = (\pi(H(B_i)), EpochId, \pi(H(B_{i-1})), \pi(EDS(B_i)), \pi(EDS_k(B_{i1})))$ of each block, there are two epoch identifiers $EpochId_{i-1}$ and $EpochId_i$, because block signatures from two consecutive epochs are required.

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

Proposals for the organization of the track at the Zero Knowledge Proofs hackathon have been developed

We offer simple tasks, which can be conditionally divided into three groups:

1. **Rust programming basics (optional).** The goal is to get acquainted with the principles of programming in the Rust language, in particular: the type system; memory management; syntax; object system; parallelism, etc. Students are offered to perform the simplest tasks related to sorting and searching for elements in arrays; recursive algorithms; combinatorics, etc. Upon successful completion, students will also implement the simplest algorithms of number theory: Euclid's algorithm; solution of the system of comparisons; modular arithmetic; basics of cryptography.
2. **Simple zkSNARKs in Rust.** The goal is to get familiar with the Rust framework when programming the simplest zkSNARKs: generating circuits, creating public settings for provers and verifiers; proof generation; proof verification. The task is to write the protocol of interaction between the verifier and the prover, describe how the data transfers between them. Students are proposed to solve the task of proving the computational integrity of hashing and digital signature algorithms. The competitive elements are the proof generation with the highest verification speed or with the smallest proof size.
3. **Aggregation and recursive zkSNARKs in Rust.** The goal is to deepen your understanding of zkSNARKs: aggregation of multiple proofs; recursive proofs; optimization by speed/size criteria. Students are asked to complete the task of aggregating multiple zkSNARKs (multiple hashes, multiple signatures; multiple hashes and signatures combined together), as well as the task of recursively calling proofs within other proofs. The complex tasks are to optimize proof generation and reach a high speed of verification and acceptable proof size at the same time.