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

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

INTRODUCTION

There is a well-known method for verifying the correctness of blocks in a blockchain called native registry check or recalculation, in other words. This method requires a significant amount of time. For instance, if we take a millionth block, it is necessary to calculate a million hashes and a million signatures. It is really computationally hard.

Thus, the main problem of blockchain projects is scalability.

The NEAR project partially solves the problem of scalability through segmentation (sharding). However, this is a temporary solution. This does not solve the problem globally, especially in the context of the mass introduction of blockchain systems.

In our research, we use ZK-SNARK technology. 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 $O(log2(t))$ 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.

Since zero-knowledge feature is not required to create proofs, from the whole ZK-SNARK technology we decided to use only succinctness and non-interactiveness.

During our research, we have developed a scheme for recursive proof of the computational integrity of a chain of linked blocks.

The scheme uses a linked list (chain of blocks) built through a cryptographic link (hashing). For each block, we form proofs for computational integrity of hashes and digital signatures, included in the block.

While implementing a proving scheme, our team decided to divide the research into two directions.

The first one, called **aggregation**. For this scheme, we create a chain of linked hashes. We generate proofs for hash and digital signature verification and then aggregate them into one proof. A chain of proofs is created by aggregating the proof of the previous block with

the current one. As a result, for each specific block, we can check the proof that the block hash and signatures were calculated correctly and the proof chain for previous blocks was verified correctly. This method requires digital signatures to confirm computational integrity of the block, otherwise a possible attacker can simply substitute some blocks.

The second idea, called **recursion**, is based on the recursively linked chain. That is, we create a genesis block. After that, we generate a single proof for hash and signatures of the current block, not aggregating them, just making an enlarged circuit. A proof of the next block is formed by verifying the previous block (i.e., by verifying the correctness of hash and signatures of the previous block) and calculating a proof for the current block. This scheme allows checking the previous hash so that we solve the block substitution problem. The scheme does not require a digital signature to provide the computational integrity of a chain.

SECTION II: Test case #1. Generating a chain of proofs of the computational integrity of a chain of linked hashes

In blockchain, each next block is formed using a cryptographic connection organized by hashing (Fig. 1).

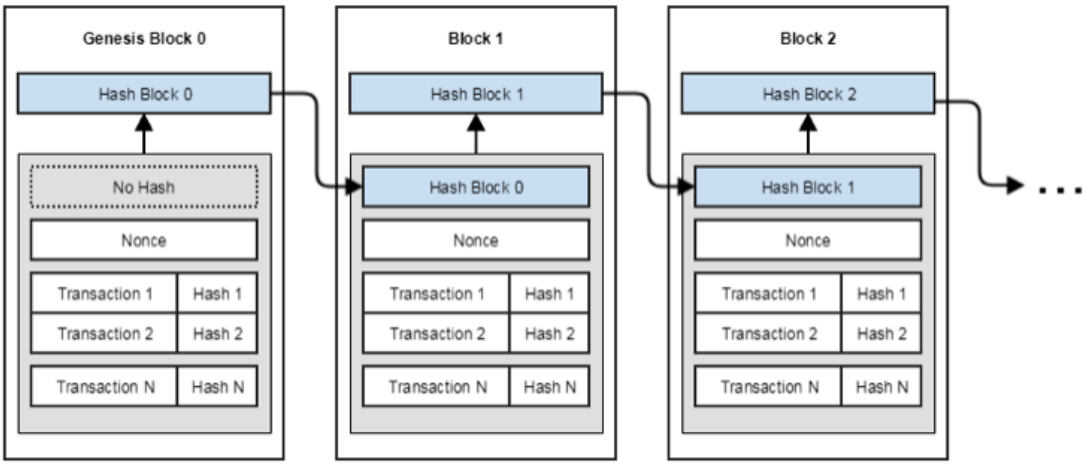

Fig. 1 — Linked list (chain of blocks) built using hashing (cryptographic link)

There are several transactions in each block, a unique number, as well as a hash of the previous block. The contents of the block (including the hash of the previous block) are hashed, and the resulting hash image is included in the next block. It is computationally difficult to find a preimage (change the contents of a block) for a known hash (not available on modern computers), which ensures reliable linking of blocks into a chain. In other words, it is computationally unattainable to change historical records in a way that does not break the integrity of the chain — changing even one bit in the contents of the block will lead to a 50% change in the hash image, which will be revealed during the integrity check.

Initial data. For a test example, let us consider a simplified version (fig. 2), when each block contains only two fields: the hash of the previous block h_{i-1} and the unique block number *Nonce*.

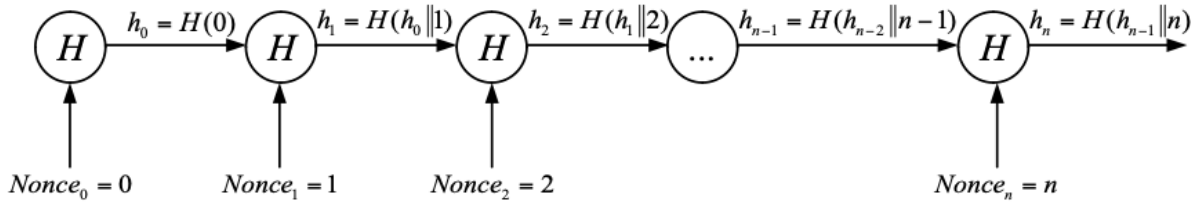

Fig. 2 — Simplified Linked List

Let us take $Nonce_i = i$. Then for each *i-th* block, hashing $h_i = H(h_{i-1}||i)$ is carried out over the result of concatenation of values h_{i-1} and i. Thus, the result of the *n-th* hashing is:

$$h_n = H(h_{n-1}\|n) = H(H(h_{n-2}\|n-1)\|n) = \ldots = H(H(H(\ldots H(H(0)\|1)\ldots\|n-2)\|n-1)\|n) \quad (1)$$

The task is to implement a test case of recursive proof of computational integrity (CI) for a chain of linked hashes. In other words, it is necessary to prove CI of h_n by the expression above.

To do this, it is necessary to consistently implement the following tasks:

1. Implement hash chain $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,\ldots n$;
2. Create a circuit $C(x_i, w_i)$ for hashing algorithm H for each $h_0,\ldots h_n$, $i = 1,\ldots n$, where $x_i = h_i$ is the result of hashing, w_i are the input data (witness, hash-preimage) to calculate hashes: $w_0 = 0$, $w_i = (h_{i-1}||i)$, $i = 1,\ldots n$;
3. Form public settings for all $i = 0,\ldots n$: $(S_{pi}, S_{vi}) = S(C_i(x_i, w_i))$, where S_{pi} are public prover settings, S_{vi} are public verifier settings;
4. Form a proof of CI for all $i = 0,\ldots n$: $\pi_i = P(S_{pi}, x_i, w_i)$;
5. Implement proof verification algorithm $V(S_{vi}, x_i, \pi_i)$ takes values *{0, 1} (accept or reject)*;
6. Proof verification, i.e., to make sure that $V(S_{vi}, x_i, \pi_i) = accept$ for all $i = 0,\ldots n$.

The result of completing tasks 1-6 is the formation of a set of proofs π_i ($i = 0,\ldots n$) of computational integrity, which is shown in figure 3.

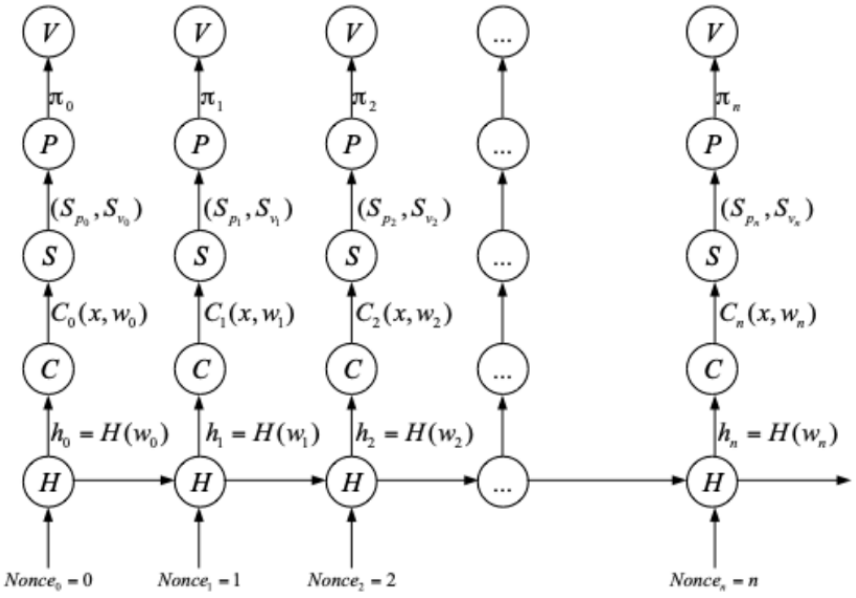

Fig. 3 — Scheme for generating a set of proofs of computational integrity

SECTION III: Test case #1. Aggregation & Recursion methods

Aggregation. To create a chain of linked hashes, i.e. to prove to the verifier V the correctness of the calculation of $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,...n$, we consider the values $\pi_i = P(S_{pi}, x_i, w_i)$ $(i = 0,...n)$ as input data W_i when aggregating proofs.

1. Form a computational scheme $C_i(X_i, W_i)$ of the verification algorithm V for each pair of proofs $\pi_{i-1} = P(S_{pi-1}, x_{i-1}, w_{i-1})$ and $\pi_i = P(S_{pi}, x_i, w_i)$, where $X_i = (V(S_{vi-1}, x_{i-1}, \pi_{i-1}), V(S_{vi}, x_i, \pi_i))$ are the results of verification (accept or reject) of proofs π_{i-1} and π_i. $W_i = (\pi_{i-1}, h_{i-1}, \pi_i, h_i)$ are the input data for verification of proofs;
2. Generate public settings $(S_{Pi}, S_{Vi}) = S(C_i(X_i, W_i))$, $(i = 1,...n)$, where S_{Pi} are public prover settings, S_{Vi} are public verifier settings;
3. Generate a proof of computational integrity $\prod_i = P(S_{Pi}, X_i, W_i)$ for all $i = 1,...n$;
4. Implement proof verification algorithm $V(S_{Vi}, X_i, \prod_i)$ takes values *{0, 1} (accept or reject)*;
5. Proof verification, i.e. to make sure that $V(S_{Vi}, X_i, \prod_i) = accept$.

Thus, each proof $\prod_i = P(S_{Pi}, X_i, W_i)$ is an aggregation of two other proofs $\pi_{i-1} = P(S_{pi-1}, x_{i-1}, w_{i-1})$ and $\pi_i = P(S_{pi}, x_i, w_i)$, which are provided as inputs (witnesses) of the proof $\prod_i$.

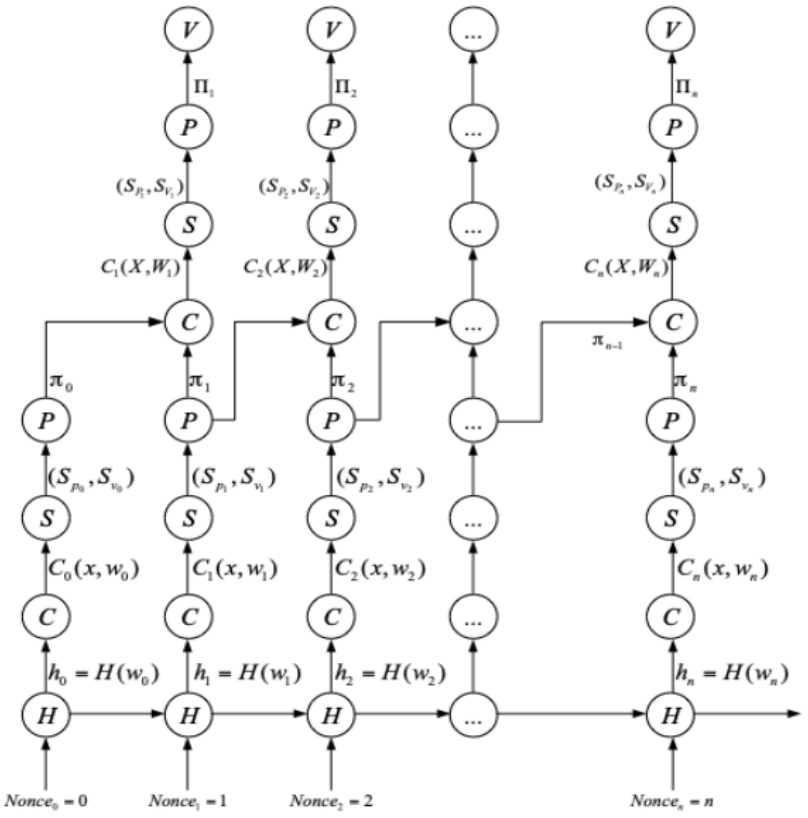

Fig. 4 — Scheme of forming a chain of recursive proofs of CI

Thus, the computational scheme $C_i(X_i, W_i)$ is constructed in such way that the result of the verification is $V(S_{Vi}, X_i, \prod_i) = V(S_{vi-1}, x_{i-1}, \pi_{i-1})$ takes values $V(S_{vi}, x_i, \pi_i)$.

Note: if the results X_i of proof verification $V(S_{vi-1}, x_{i-1}, \pi_{i-1}) = accept$ and $V(S_{vi}, x_i, \pi_i) = accept$ are directly used in the formation of a computational scheme, then only witness W_i can be fed to the input of the scheme, i.e. the scheme looks like: $C_i(W_i)$ without public inputs X_i.

Fig. 5 shows another version of the recursive proof scheme, in which:

1. $X_i = (V(S_{vi-1}, x_{i-1}, \pi_{i-1}), V(S_{vi}, x_i, \pi_i))$ for all $i = 1,...n$.
2. $W_1 = (\pi_0, h_0, \pi_1, h_1)$, $W_i = (\prod_{i-1}, X_{i-1}, \pi_i, h_i)$ for all $i = 2,...n$.

In this case, the fulfillment of the condition $V(S_{Vi}, X_i, \prod_i) = accept$ for all $i = 2,...n$ means that proof verification $V(S_{Vi-1}, X_{i-1}, \prod_{i-1})$ and $V(S_{vi}, x_i, \pi_i)$ were calculated correctly. If at the same time $V(S_{Vi-1}, X_{i-1}, \prod_{i-1}) = accept$ and $V(S_{vi}, x_i, \pi_i) = accept$, this means that:

1. *Proof verification V(SVi-2, Xi-2, ∏i-2) and V(Svi, xi, πi) were calculated correctly;*
2. *The value hi = H(hi-1||i) is calculated correctly.*

Thus, the second chain example of recursive proofs (Fig. 5) aggregates in each $\prod_i$ all previous proofs ($i = 0,...n$). For instance, the last proof in the chain $\prod_n = P(S_{Pn}, X_n, W_n)$ verifies the correctness of value $h_n = H(h_{n-1}||n)$ and all previous proofs.

To form the proof $\prod_n = P(S_{Pn}, X_n, W_n)$, the following are used as input data:

1. *Xn = (V(Svn-1, xn-1, πn-1), V(Svn, xn, πn)) are the results of verification of two previous proofs;*
2. *Wn = (∏n-1, Xn-1, πn, hn), i.e. the proof ∏n-1 and the result of its verification and the proof πn.*

Conditions $V = accept$ should be used by forming all computational schemes of recursive proofs. Then the result of the verification $V(S_{Vi}, X_i, \prod_i) = accept$ means that all hashes in the chain (Fig. 1) are calculated correctly and all verifications for all $i = 0,...n$ are also calculated correctly.

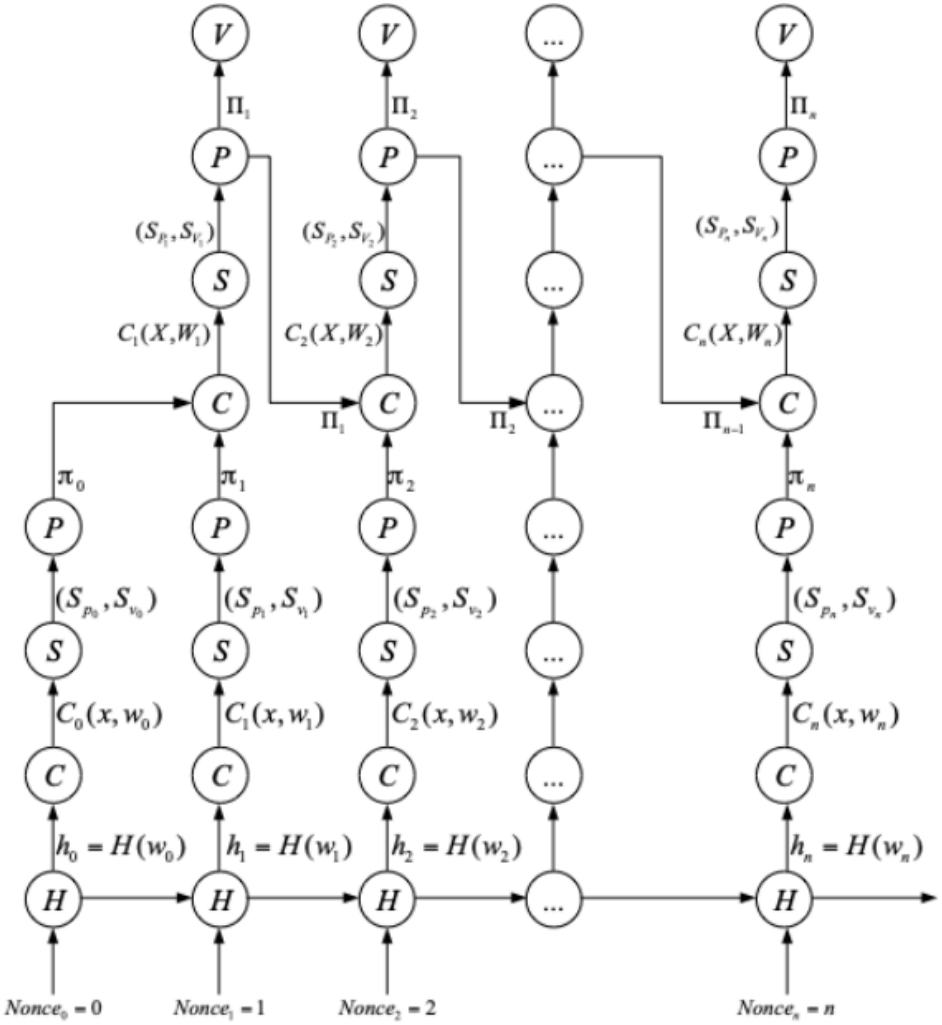

Fig. 5 — Scheme of forming a chain of recursive proofs of computational integrity

Recursion. To create a chain of linked hashes $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,...n$, we consider the values $\pi_i = P(S_{pi}, x_i, w_i)$ $(i = 0,...n)$ as input data W_i when creating a proof for a next block.

1. Form a computational scheme $C_i(X_i, W_i)$, which includes the verification algorithm V for the previous block $\pi_{i-1} = P(S_{pi-1}, x_{i-1}, w_{i-1})$ and hashing algorithm H for the current block, where $X_i = (V(S_{vi-1}, x_{i-1}, \pi_{i-1}), x_i)$: $V(S_{vi-1}, x_{i-1}, \pi_{i-1})$ is the result of verification (accept or reject) of the previous proof π_{i-1} and $x_i = h_i$ is the result of hashing. $W_i = (\pi_{i-1}, h_{i-1}, w_i)$: π_{i-1}, h_{i-1} are the input data for verification of proof and w_i are the input data (witness, hash-preimage) to calculate hashes: $w_0 = 0$, $w_i = (h_{i-1}||i)$, $i = 1,...n$;
2. Generate public settings $(S_{Pi}, S_{Vi}) = S(C_i(X_i, W_i))$ $(i = 1,...n)$, where S_{Pi} are public prover settings, S_{Vi} are public verifier settings;
3. Generate a proof of computational integrity $\pi_i = P(S_{Pi}, X_i, W_i)$ for all $i = 1,...n$;

4. Implement proof verification algorithm $V(S_{Vi}, X_i, \pi_i)$ takes values *{0, 1} (accept or reject)*;
5. Proof verification, i.e. to make sure that $V(S_{Vi}, X_i, \pi_i) = accept$.

Thus, each proof $\pi_i = P(S_{Pi}, X_i, W_i)$ is formed, based on the previous one $\pi_{i-1} = P(S_{pi-1}, x_{i-1}, w_{i-1})$ and its own hashing algorithm.

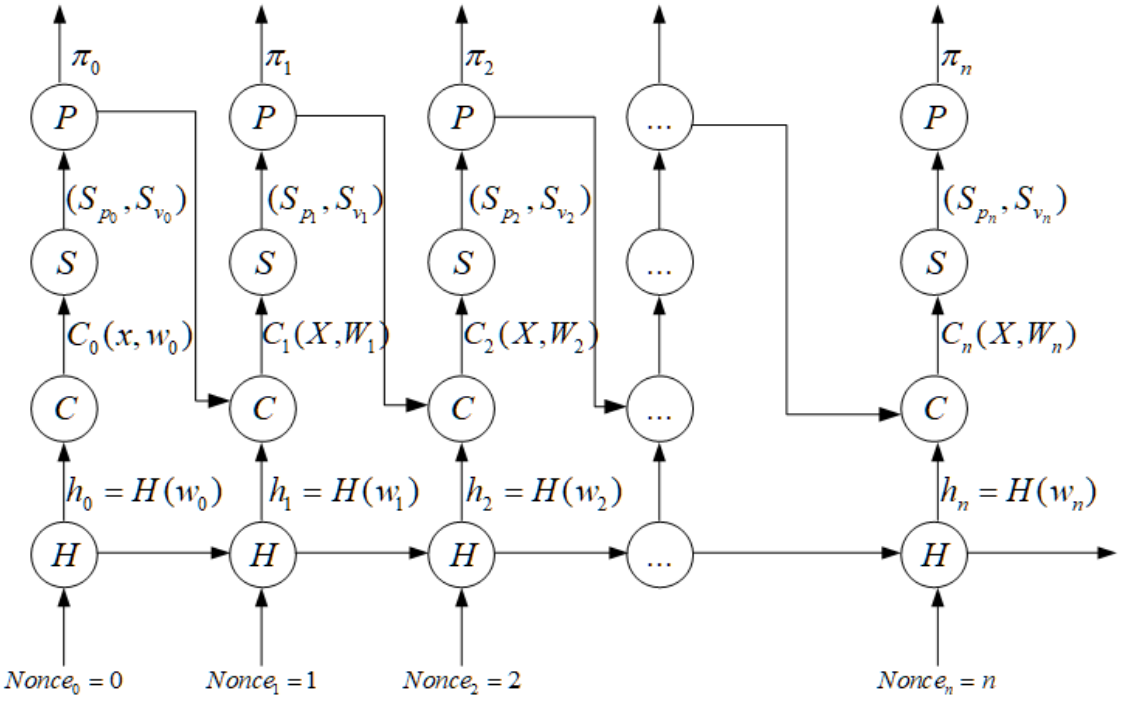

Fig. 6 — Scheme of forming a chain of recursive proofs of computational integrity

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

SECTION IV: Test case #1. Time and measurement results

All computations were made on the 1,8 GHz Intel Core i5 and 1,900GHz AMD Ryzen 7 5800U (16) for comparison.

Aggregation. The scheme implementation:

https://github.com/tikhono/zkp-research/tree/proof_to_file/plonky2_aggregation

Table 1 — Time and measurement results for generating separate blocks (1,8 GHz Intel Core i5)

№	Hash	Time to build a separate circuit, s	Time to make a separate proof, s	Proof size (aggregated with the previous one), bytes
0	5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9	2.289142	4.1930	132640
1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	1.9941112	4.0253	132816
2	9C808663A3D1A47F3FCFE26BA11AFD90D0F00A41984F8058DE3CE8552C6BEF77	2.3753016	4.2972	132816
3	B3CC82D30374FC4FA584032CF35E86A54E70E08A9E3E9982BFE06A6022A4937A	2.2045727	4.3221	132816
4	4D23AF03289C3EB09E4C888999102EE53E6A49D49FD06326A09ED924D22D34A9	2.0633605	3.9754	132816

Table 2 — Time and measurement results for a chain of aggregated proofs (1,8 GHz Intel Core i5)

№	Aggregation				Time to build a circuit, s	Time to prove, s	Time to verify, s
0					2.0916486	4.4853	0.0239
1					2.2827702	4.7302	0.0214

2					2.4975781	4.3795	0.0282
3					2.2045841	4.3959	0.0211
4							

Table 3 — Time and measurement results for generating separate blocks (1,900GHz AMD Ryzen 7 5800U (16))

№	Hash	Time to build a separate circuit, s	Time to make a separate proof, s	Proof size (aggregated with the previous one), bytes
0	5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9	0.51996005	0.7165	132640
1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	0.45409113	0.6317	132816
2	9C808663A3D1A47F3FCFE26BA11AFD90D0F00A41984F8058DE3CE8552C6BEF77	0.4530549	0.6424	132816
3	B3CC82D30374FC4FA584032CF35E86A54E70E08A9E3E9982BFE06A6022A4937A	0.46004227	0.6430	132816
4	4D23AF03289C3EB09E4C888999102EE53E6A49D49FD06326A09ED924D22D34A9	0.5011626	0.8347	132816

Table 4 — Time and measurement results for a chain of aggregated proofs (1,900GHz AMD Ryzen 7 5800U (16))

№	Aggregation				Time to build a circuit, s	Time to prove, s	Time to verify, s
0					0.53887284	0.9314	0.0058
1					0.5421975	0.7730	0.0061
2					0.5415488	0.6894	0.0070
3					0.5190188	0.7198	0.0055

Recursion. The scheme implementation:

https://github.com/tikhono/zkp-research/tree/main/plonky2_recursion

Table 5 — Time and measurement results for a recursive chain of proofs for hashes (1,8 GHz Intel Core i5)

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	2.6360054	4.5231	0.0104	133024
1	AC8BE15C3CC494661 BC32FF3457E273A988 96338195A8127021085 74D9930EAD	5.016265	11.3599	0.0127	149084
2	3730496E35571584A28 39C243481A461AA7E3 04203666CB6A358247 CB2818914	4.6889973	8.8982	0.0118	149084
3	C9C25FCF5183B139A4 62ECD06919572A8805 9355D3271F5CFDBBC D46F32C6723	4.731547	9.8425	0.0185	149084

Table 6 — Time and measurement results for a recursive chain of proofs for hashes (1,900GHz AMD Ryzen 7 5800U (16))

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	0.5414246	0.9578	0.0063	133024

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

1	AC8BE15C3CC494661BC32FF3457E273A98896338195A812702108574D9930EAD	0.53146	0.9618	0.0065	149084
2	3730496E35571584A2839C243481A461AA7E304203666CB6A358247CB2818914	0.55626	0.9423	0.006	149084
3	C9C25FCF5183B139A462ECD06919572A88059355D3271F5CFDBBCD46F32C6723	0.541346	0.9614	0.0062	149084

SECTION V: Test case #2. Generating a chain of proofs of the computational integrity of a chain of linked hashes with digital signature

Test case #2 implements a recursive proof of the computational integrity of a chain of linked hashes with digital signature verification. This protects against possible data changes (evidence and hashes) and the imposition of a false chain of related hashes.

Initial data. For a test case, consider a simplified version from test case #1, supplemented by digital signature verification (fig. 6). Each block contains three fields:

1. Unique block number: $Nonce_i$;
2. Hash of the previous block: h_{i-1};
3. Digital signature: $EDS(h_i)$.

To simplify (as in test case #1), we accept $Nonce_i = i$. The result of the *n-th* hashing is:

$$h_n = H(h_{n-1} \| n) = H(H(h_{n-2} \| n-1) \| n) = \ldots = H(H(H(\ldots H(H(0) \| 1) \ldots \| n-2) \| n-1) \| n) \quad (2)$$

Additionally, each hash h_i is encrypted with a secret key *sk*, i.e. we form a signature $EDS(h_i, sk)$. The public key *pk* is used to verify the signature, i.e. we decrypt EDS_i and check for equality $h_i = D(EDS_i, pk)$. We use only the signature verification algorithm $h_i = D(EDS_i, pk)$ to generate proofs and CI verification.

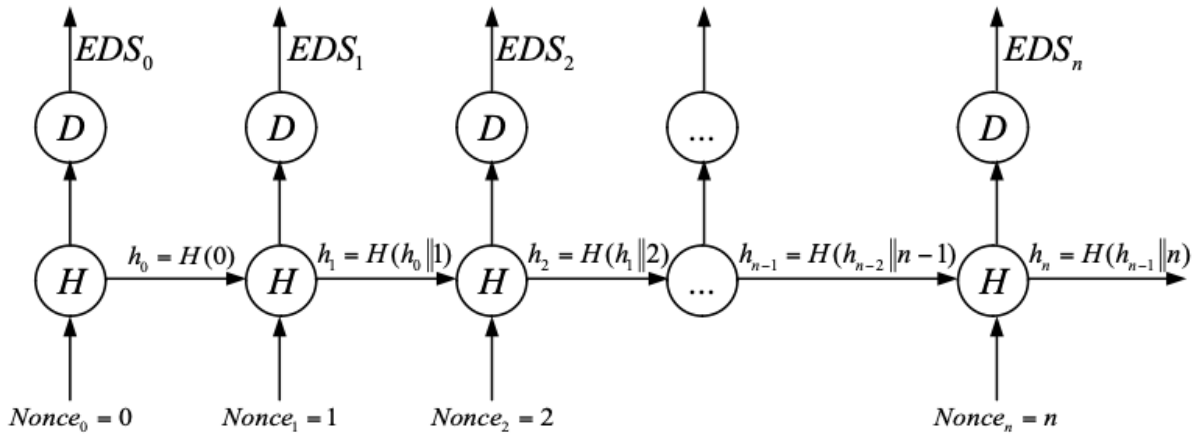

Fig. 7 — A simplified version of the linked list with the formation of EDS

SECTION VI: Test case #2. Aggregation & Recursion methods

Aggregation. To implement a recursive proof of CI of a chain, it is necessary to consistently implement the following tasks:

1. Implement a hash chain $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,...n$;
2. For each hash $h_0,... h_n$:
    a. Create a circuit $CH_i(xH_i, wH_i)$ of the hash algorithm H, where the public input $xH_i = h_i$ is the result of hashing, the witness wH_i is the hash preimage: $wH_0 = 0$, $wH_i = h_{i-1}||i$, $i = 1,...n$;
    b. Form public settings $(SH_{pi}, SH_{vi}) = S(CH_i(xH_i, wH_i))$, where SH_{pi} are public prover settings, SH_{vi} are public verifier settings;
    c. Form a proof CI for hashing $\pi H_i = P(SH_{pi}, xH_i, wH_i)$;
    d. Implement verification algorithm $V(SH_{vi}, xH_i, \pi H_i)$ takes values *{0, 1} (accept or reject)*;
    e. Proof verification, i.e. to make sure that $V(SH_{vi}, xH_i, \pi H_i) = accept$.
3. For each signature $EDS_i = E(h_i, sk)$, $i = 0,...n$:
    a. Create a circuit $CD_i(xD_i, wD_i)$ of proof verification $h_i = D(EDS_i, pk)$, where the public input $xD_i = h_i$ is the result of hashing, the witness $wD_i = (EDS_i, pk)$ are the signature and the public key;
    b. Form public settings $(SD_{pi}, SD_{vi}) = S(CD_i(xD_i, wD_i))$, where SD_{pi} are public prover settings, SD_{vi} are public verifier settings;
    c. Form a proof CI for signature verification $\pi D_i = P(SD_{pi}, xD_i, wD_i)$;
    d. Implement proof verification algorithm $V(SD_{vi}, xD_i, \pi D_i)$ takes values *{0, 1} (accept or reject)*;
    e. Proof verification, i.e. to make sure that $V(SD_{vi}, xD_i, \pi D_i) = accept$.
4. For every triple of proofs $\prod_{i-1} = P(S_{Pi-1}, X_{i-1}, W_{i-1})$, $\pi H_i = P(SH_{pi}, xH_i, wH_i)$ and $\pi D_i = P(SD_{pi}, xD_i, wD_i)$, $i = 1,...n$:
    a. Create a circuit $C_i(X_i, W_i)$ verification algorithm V, where

$X_i = (V(S_{Vi-1}, X_{i-1}, \prod_{i-1})), V(SH_{vi}, xH_i, \pi H_i), V(SD_{vi}, xD_i, \pi D_i)$, for all $i = 1,...n$.

$W_1 = (\pi_0, h_0, \pi_1, h_1, EDS_1, pk)$, $W_1 = (\prod_{i-1}, X_{i-1}, \pi_i, h_i, EDS_i, pk)$, for all $i = 2,...n$.

    b. Form public settings $(S_{Pi}, S_{Vi}) = S(C(X_i, W_i))$, where S_{Pi} are public prover settings, S_{Vi} are public verifier settings;
    c. Form a proof of CI $\prod_i = P(S_{Pi}, X_i, W_i)$;

d. Implement proof verification algorithm $V(S_{Vi}, X_i, \prod_i)$ takes values *{0, 1} (accept or reject);*

e. Proof verification, i.e. to make sure that $V(S_{Vi}, X_i, \prod_i) = accept$.

Thus, each proof $\prod_i = P(S_{Pi}, X_i, W_i)$, $i = 1,...n$ is the aggregation of three other proofs:

1. Proof CI of previous chain of linked hashes $\prod_{i-1} = P(S_{Pi-1}, X_{i-1}, W_{i-1})$;
2. Proof CI of current hash $\pi H_i = P(SH_{pi}, xH_i, wH_i)$;
3. Proof CI of current signature verification $\pi D_i = P(SD_{pi}, xD_i, wD_i)$.

Proof $\prod_0 = P(S_{P0}, X_0, W_0)$ is the aggregation of two proofs:

1. Proof CI of current hash $\pi H_0 = P(SH_{p0}, xH_0, wH_0)$;
2. Proof CI of current signature verification $\pi D_0 = P(SD_{p0}, xD_0, wD_0)$.

The scheme of forming a chain of recursive proofs of computational integrity with verification of the correctness of electronic digital signatures is shown in fig. 8.

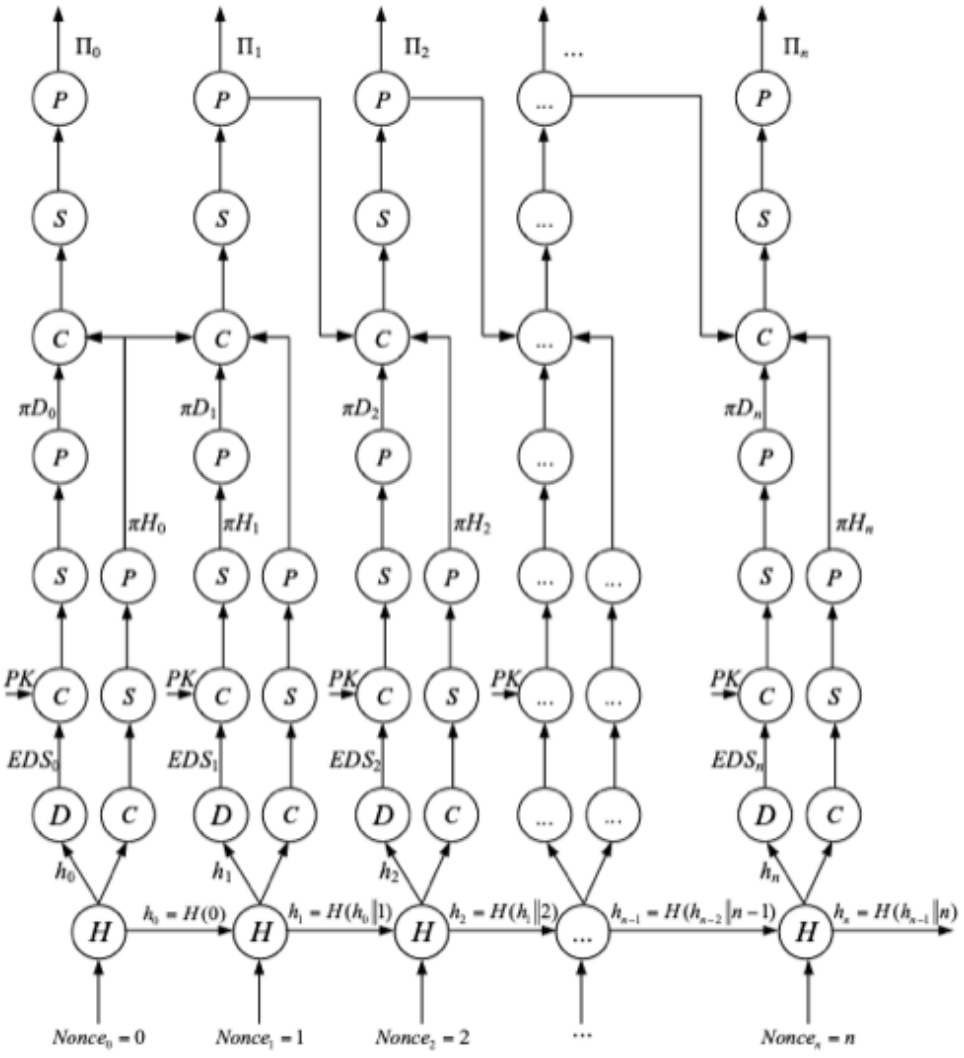

Fig. 8 — Scheme of forming a chain of proofs of CI with digital signature verification

Condition fulfillment $V(S_{Vi}, X_i, \prod_i) = accept$ for all $i = 1,...n$ means that the proof verification $\prod_{i-1} = P(S_{Pi-1}, X_{i-1}, W_{i-1})$, $\pi H_i = P(SH_{pi}, xH_i, wH_i)$ and $\pi D_i = P(SD_{pi}, xD_i, wD_i)$ were calculated correctly. If $V(S_{Vi-1}, X_{i-1}, \prod_{i-1}) = accept$, $V(SH_{vi}, xH_i, \pi H_i) = accept$ and $V(SD_{vi}, xD_i, \pi D_i) = accept$, it means that:

1. There is a proof of CI of previous chain, i.e. the verification $V(S_{Vi-2}, X_{i-2}, \prod_{i-2}) = accept$ is computed correctly;
2. There is a proof of CI of current hash, i.e. the value $h_i = H(h_{i-1}||i)$ is computed correctly;
3. There is a proof of CI of current signature, i.e. verification $h_i = D(EDS_i, pk)$ is computed correctly.

Recursion. To implement a recursive proof of CI of a chain, it is necessary to consistently implement the following tasks:

1. Implement a block chain based on linked hashes $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,...n$;
2. For each block create a circuit $C_i(X_i, W_i)$ of the hash algorithm H, signature verification and the verification algorithm V for the previous block $\pi_{i-1} = P(S_{pi-1}, x_{i-1}, w_{i-1})$. $X_i = (V(S_{vi-1}, x_{i-1}, \pi_{i-1}), x_i)$: $V(S_{vi-1}, x_{i-1}, \pi_{i-1})$ is the result of verification (accept or reject) of the previous proof π_{i-1}, $x_i = h_i$ is the result of hashing. $W_i = (\pi_{i-1}, h_{i-1}, wH_i, wD_i)$: π_{i-1}, h_{i-1} are the input data for verification of proof and w_i are the input data (witness, hash-preimage) to calculate hashes: $w_0 = 0$, $w_i = (h_{i-1}||i)$, $i = 1,...n$, the witness $wD_i = (EDS_i, pk)$ are the signature and the public key;
3. Form public settings $(S_{pi}, S_{vi}) = S(CH_i(xH_i, wH_i), CD_i(xD_i, wD_i))$, where S_{pi} are public prover settings, S_{vi} are public verifier settings;
4. Implement proof verification algorithm $V(S_{vi}, x_i, \pi_i)$ takes values *{0, 1} (accept or reject)*;
5. Proof verification, i.e. to make sure that $V(S_{vi}, x_i, \pi_i) = accept$.

Condition fulfillment $V(S_{Vi}, X_i, \prod_i) = accept$ for all $i = 1,...n$ means that there is a proof of CI of previous chain, i.e. the verification $V(S_{vi-1}, x_{i-1}, \pi_{i-1}) = accept$ is computed correctly, CI of current hash, i.e. the value $h_i = H(h_{i-1}||i)$ is computed correctly and CI of current signature, i.e. verification $h_i = D(EDS_i, pk)$ is computed correctly.

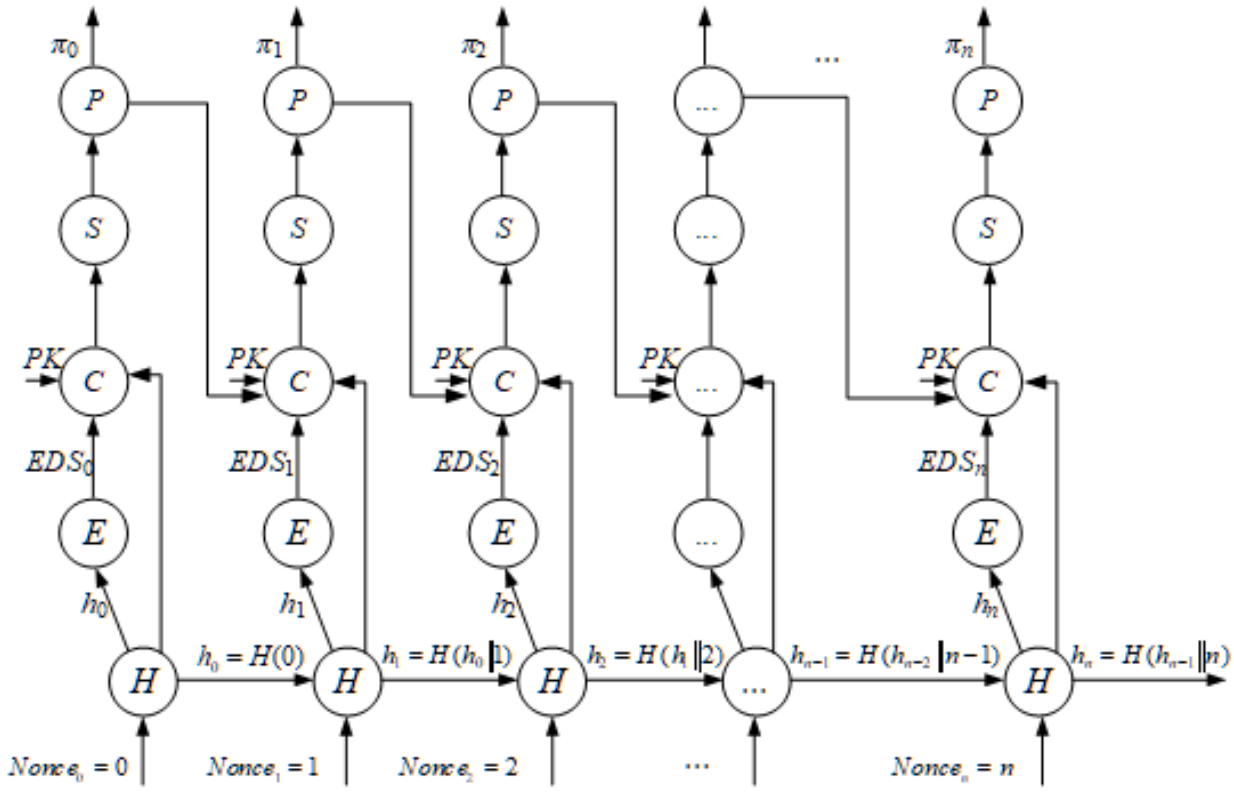

Fig. 9 — Scheme of forming a chain of proofs of CI with digital signature verification

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

SECTION VII: Test case #2. Time and measurement results

All computations were made on 1,900GHz AMD Ryzen 7 5800U (16) and 2,40GHz Intel(R) Xeon(R) CPU E5-2680 v4 for comparison.

Aggregation. The scheme implementation:

https://github.com/tikhono/zkp-research/tree/proof_to_file/plonky2_sig_hash

Table 7 — Time and measurement results for a chain of proofs for hashes (2,40GHz Intel(R) Xeon(R) CPU E5-2680 v4)

№	Hash	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9	1.111688	0.9969	132640	0.0224
1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	2.2134259	1.6811	150268	0.0282
2	9C808663A3D1A47F3FCFE26BA11AFD90D0F00A41984F8058DE3CE8552C6BEF77	2.2696147	1.7133	150268	0.1521
3	B3CC82D30374FC4FA584032CF35E86A54E70E08A9E3E9982BFE06A6022A4937A	2.3223338	1.7160	150268	0.1571
4	4D23AF03289C3EB09E4C888999102EE53E6A49D49FD06326A09ED924D22D34A9	2.8574667	2.6339	150268	0.0621

Table 8 — Time and measurement results for a chain of proofs for signatures (2,40GHz Intel(R) Xeon(R) CPU E5-2680 v4)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	63.335045	75.2214	208376	0.0170
1	58.583714	61.1627	208376	0.0173
2	64.45872	57.5522	208376	0.0225
3	71.951866	72.8148	208376	0.0182
4	102.80682	59.8327	208376	0.0168

Table 9 — Time and measurement results for a chain of aggregated proofs for signatures & hashes (2,40GHz Intel(R) Xeon(R) CPU E5-2680 v4)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	2.6056433	1.4931	146348	0.0122
1	2.6340964	1.7561	146348	0.0120
2	2.5258436	1.6699	146348	0.0171
3	3.8990123	2.5616	146348	0.0178
4	2.6605875	1.7653	146348	0.0138

Table 10 — Time and measurement results for a chain of proofs for hashes (1,900GHz AMD Ryzen 7 5800U (16))

№	Hash	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	5FECEB66FFC86F38 D952786C6D696C79C 2DBC239DD4E91B46 729D73A27FB57E9	0.529693	0.6954	132640	0.0104

1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	1.0661488	1.7893	150268	0.0152
2	9C808663A3D1A47F3FCFE26BA11AFD90D0F00A41984F8058DE3CE8552C6BEF77	1.0135794	2.0026	150268	0.1044
3	B3CC82D30374FC4FA584032CF35E86A54E70E08A9E3E9982BFE06A6022A4937A	1.0293587	2.4225	150268	0.0471
4	4D23AF03289C3EB09E4C888999102EE53E6A49D49FD06326A09ED924D22D34A9	1.0987031	1.5323	150268	0.0957

Table 11 — Time and measurement results for a chain of proofs for signatures (1,900GHz AMD Ryzen 7 5800U (16))

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	32.144226	72.0099	208376	0.0117
1	31.208908	94.6699	208376	0.01
2	29.096764	103.2076	208376	0.0105
3	30.003609	98.6119	208376	0.02
4	31.844381	69.1603	208376	0.0127

Table 12 — Time and measurement results for a chain of aggregated proofs for signatures & hashes (1,900GHz AMD Ryzen 7 5800U (16))

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	1.3389456	1.6592	146348	28

1	1.4501207	1.6903	146348	0.0358
2	1.5479413	1.8390	146348	0.0249
3	1.3872194	2.3278	146348	0.0251
4	1.3421979	1.6491	146348	0.0063

Recursion. The scheme implementation:

https://github.com/tikhono/zkp-research/tree/main/plonky2_recursion

Table 13 — Time and measurement results for a chain of recursive proofs (2,40GHz Intel(R) Xeon(R) CPU E5-2680 v4)

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	1.119945	1.4125	0.0213	149084
1	AC8BE15C3CC494661 BC32FF3457E273A988 96338195A8127021085 74D9930EAD	72.49396	74.2432	0.0457	211432
2	3730496E35571584A28 39C243481A461AA7E3 04203666CB6A358247 CB2818914	53.94758	46.4669	0.0337	211432
3	C9C25FCF5183B139A4 62ECD06919572A8805 9355D3271F5CFDBBC D46F32C6723	50.751015	48.2629	0.0339	211432
4	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	52.462822	44.4447	0.0379	211432

Table 14 — Time and measurement results for a chain of recursive proofs (1,900GHz AMD Ryzen 7 5800U (16))

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	0.5414246	0.9578	0.0063	149084
1	AC8BE15C3CC494661 BC32FF3457E273A988 96338195A8127021085 74D9930EAD	29.565964	78.2291	0.0174	211432
2	3730496E35571584A28 39C243481A461AA7E3 04203666CB6A358247 CB2818914	29.53088	59.7240	0.0192	211432
3	C9C25FCF5183B139A4 62ECD06919572A8805 9355D3271F5CFDBBC D46F32C6723	30.960339	74.2933	0.0382	211432
4	81DDC8D248B2DCCD D3FDD5E84F0CAD62 B08F2D10B57F9A831C 13451E5C5C80A5	32.037373	84.5603	0.0279	211432

SECTION VIII: 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.

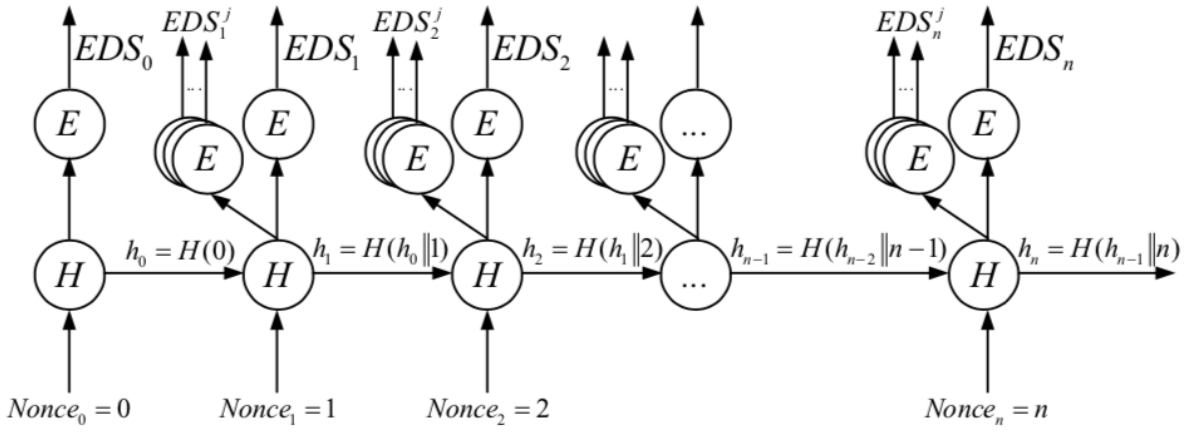

Fig. 10 – A simplified version of the linked list with digital signatures of all producers

SECTION IX: 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).

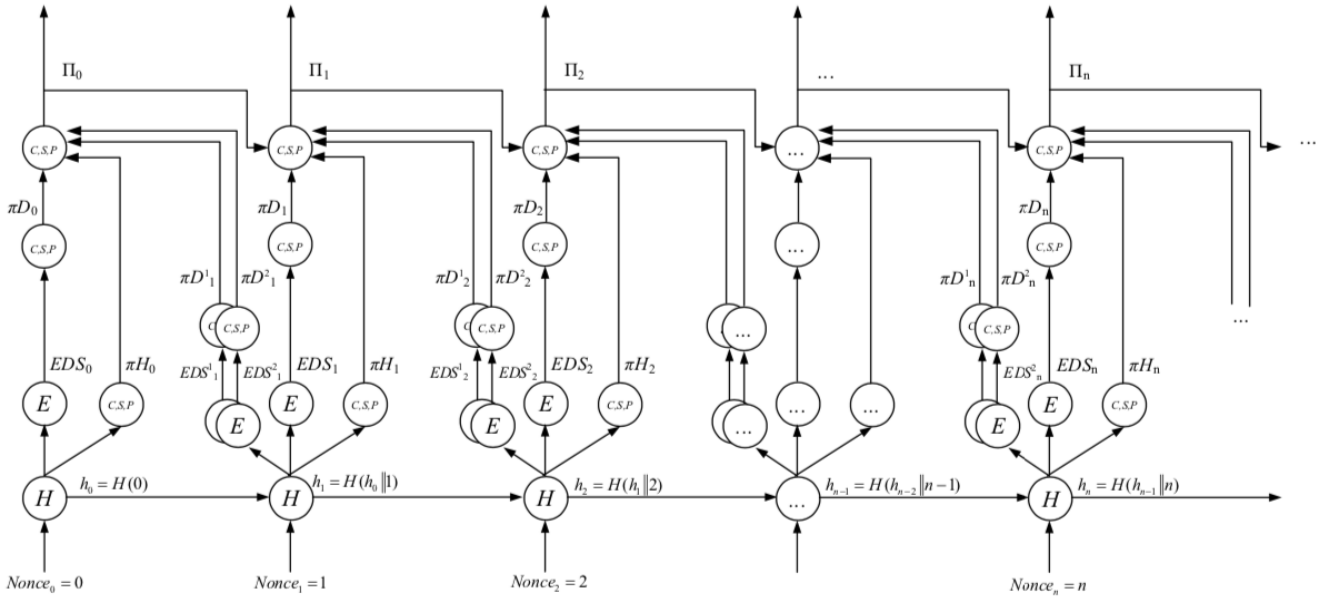

Fig. 11 – Scheme of formation of a chain of recursive proofs of CI with verification of the correctness of digital signatures of all producers

SECTION X: 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 15 — Time and measurement results for a chain of 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	2.0148926	3.8952	132640	0.0460
1	F3C1D27925BF4262AE19BFCBCCE1B76124F54A4DBF62DC4B09E6B353F34D277D	5.3175316	8.3791	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 16 — Time and measurement results for a chain of 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 17 — Time and measurement results for a chain of 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 10 — Time and measurement results for two other producers 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	88.897575	2130.3268	208376	0.3173
1	90.458694	1973.6750	208376	0.1916

2	92.04736	1654.5305	208376	0.1788
3	96.08491	2369.9976	208376	0.0587
4	92.45214	2132.1145	208376	0.0988

Table 11 — Time and measurement results for a final aggregated proof (1,8 GHz Intel Core i5)

№	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	15.812389	18.7058	152684	0.1192
1	13.8023624	19.8745	152684	0.0966
2	14.623742	19.9798	152684	0.1382
3	14.811343	18.8217	152684	0.0967
4	15.231551	18.1897	152684	0.0928