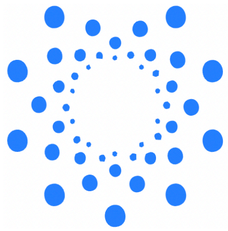
Recursive Proofs for NEAR:
Solving the Scalability Problem



ZPOKEN

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141
https://zpoken.io/
support@zpoken.io

Blockchain scalability problem

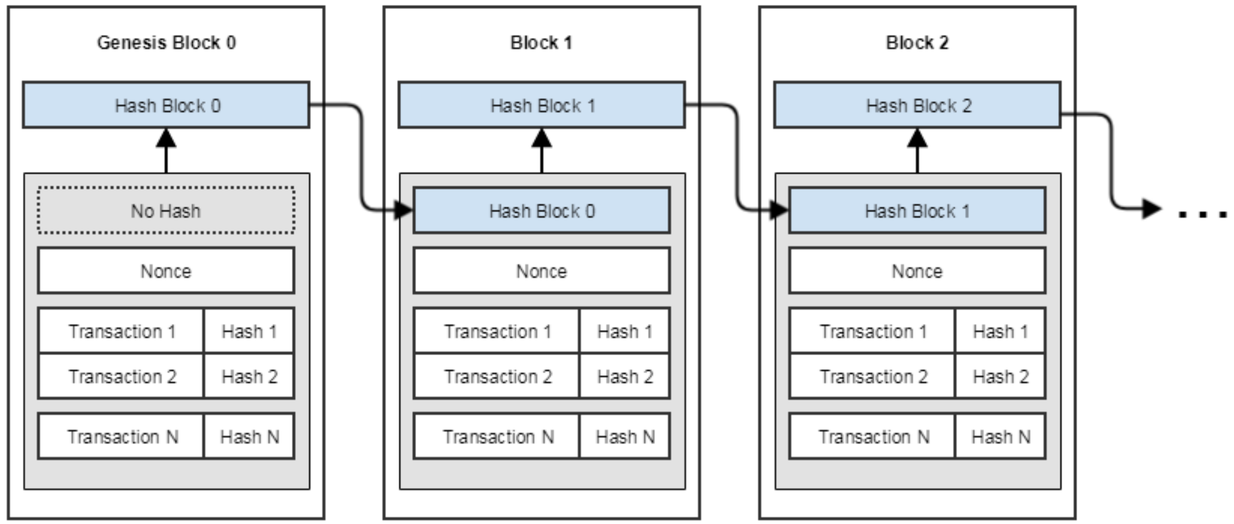
Scalability is the property of a system to continue to evolve as the amount of work increases.

Blockchain is a form of distributed ledger (DLT). DLT is formed by adding information to a continuous chain of linked blocks. Connection between blocks is provided by hashing. Additionally, each block contains digital signatures that ensure the authenticity of the chain.

Native registry check is recalculation, i.e. to recreate a chain of linked hashes and verify all signatures. This is extremely long.

Thus, the main problems of blockchain projects are ***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.



ZK-SNARK: Solving the scalability problem

ZK-SNARK is an acronym that stands for “Zero-Knowledge Succinct Non-Interactive Argument of Knowledge.”

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

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. As an example let us take a binary logarithm, to check a millionth proof we need approximately $20^2 \approx 400$ steps, which is significantly less than the native registry check, which requires $2^{20} \approx 1$ million steps. So this solves the scalability problem.

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

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

ZK-SNARK: Solving the scalability problem

Prover time is nearly linear in T

&

Verifier time exponentially smaller than T

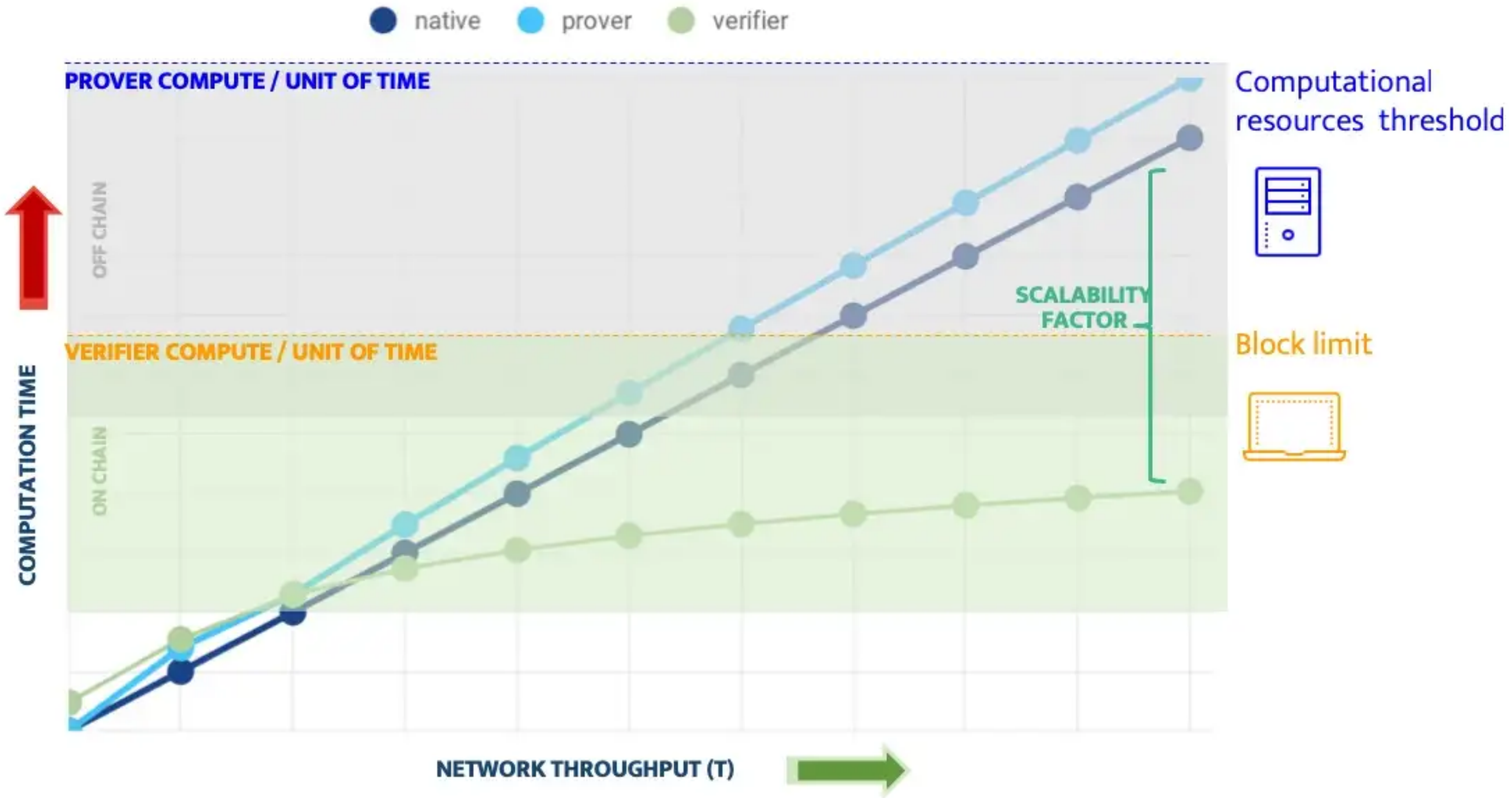


Fig. 1 — Computational complexity of native check, proof generation and verification

Tasks we solve

We want to create a prototype (MVP) of a system based on the ZK-SNARK technology that solves the problem of scalability and confidentiality of the NEAR blockchain project:

- Implement proving and verifying procedures for the main crypto-primitives (hashing and digital signature) as ZK-SNARK circuits;
- Implement a recursive proof generation and verification for the main crypto-primitives;
- Implement a proof generation and verification for a chain of recursive hash proofs;
- Implement a proof generation and verification for a chain of recursive proofs of hashes and digital signatures;
- Implement the elements of the NEAR protocol. Implement a proof generation and verification for a chain of recursive proofs of hashes and digital signatures with imitation of the NEAR protocol.

Our solutions and results. Test case #1

Let us consider a simplified version, when each block contains only two fields: the hash of the previous block h_{i-1} and the unique block number *Nonce*.

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

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

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* by the expression above.

Our solutions and results. Test case #1

Since we need to generate a set of proofs π_i $(i = 0,\ldots n)$ of computational integrity, which is shown in Figure 1.

The values $\pi_i = P(S_{pi}, x_i, w_i)$, where S_{pi} are public prover settings, $x_i = h_i$ is the result of hashing, w_i are the input data (witness, hash-preimage) to calculate hashes: $w_0 = 0$, $w_i = (h_{i-1}||i)$, $i = 1,\ldots n$.

They allow the verifier V to check the correctness of the calculation of $h_0 = H(0)$, $h_i = H(h_{i-1}||i)$, $i = 1,\ldots n$.

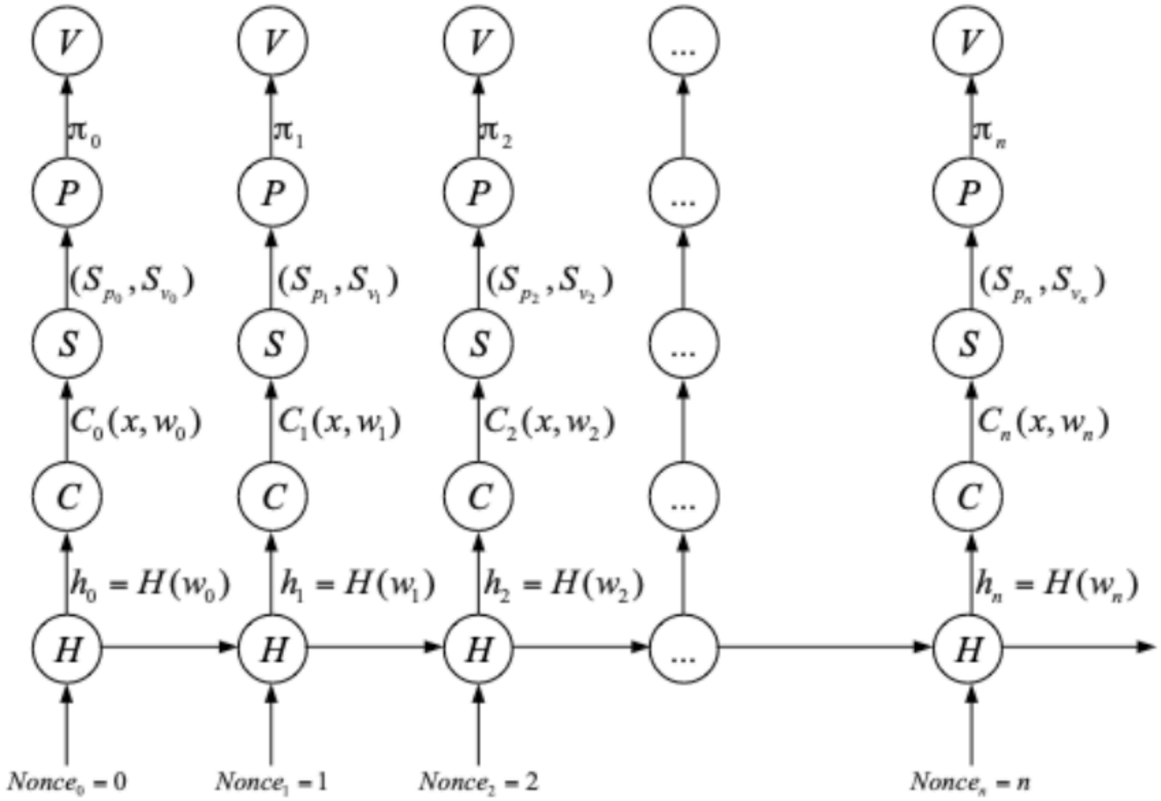


Fig. 2 — Scheme for generating a set of proofs of computational integrity

Our solutions and results. Test case #1: Aggregation[ref]

The following scheme (Fig. 2) aggregates in each $\prod_i$ all previous proofs *(i = 0,…n)*. The last proof in the chain $\prod_n = P(S_{Pn}, X_n, W_n)$ can be used to verify the correctness of value $h_n = H(h_{n-1}||n)$.

This scheme provides proof of CI of a chain of linked hashes. However, it does not consider the possibility of substituting one or more blocks. The scheme does not provide proof that the digital signature was calculated correctly.

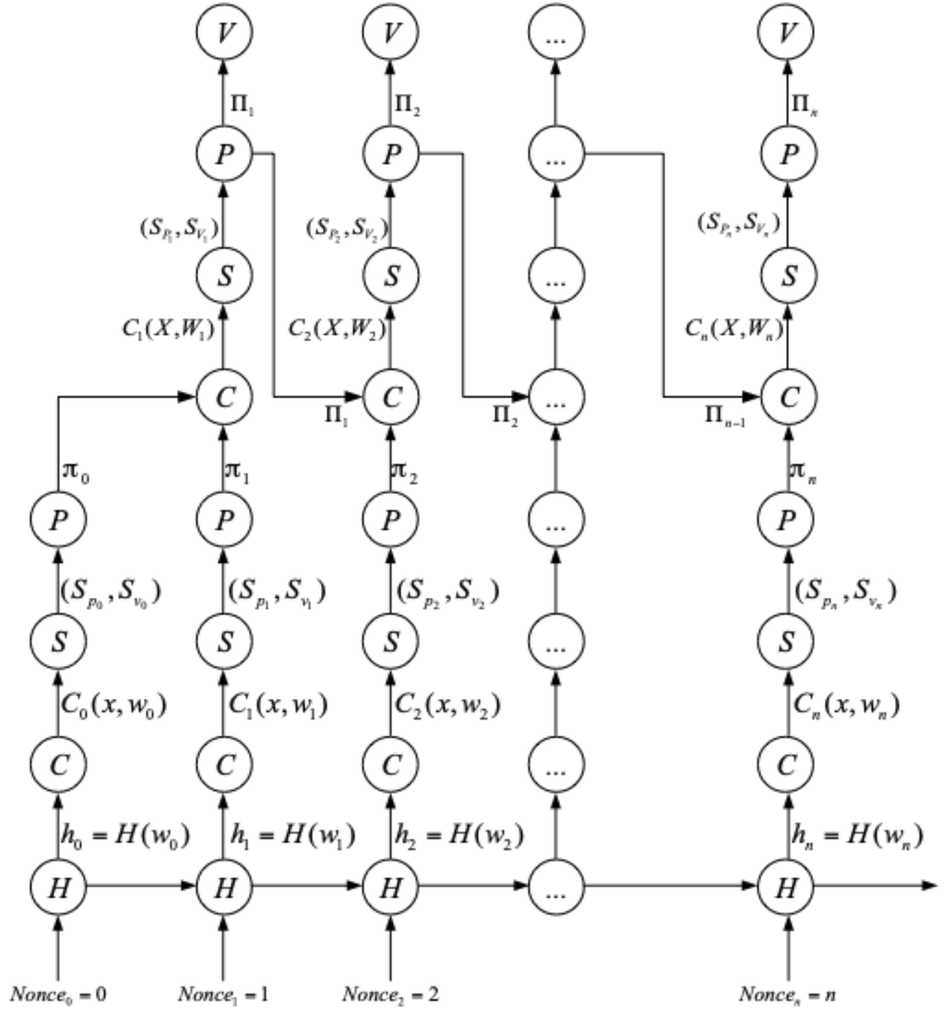


Fig. 3 — Scheme of forming a chain of recursive proofs of computational integrity

Our solutions and results. Test case #1: Recursion[ref]

The second idea is based on the recursively linked chain. That is, we create a genesis block (hash) and then a proof. A proof of the next block is formed by verifying the previous block (i.e. by verifying the hash of the previous block) and calculating a proof for the current block.

This scheme allows verification of the previous hash so that we solve the block substitution problem. The scheme does not need a digital signature to provide the computational integrity of a chain.

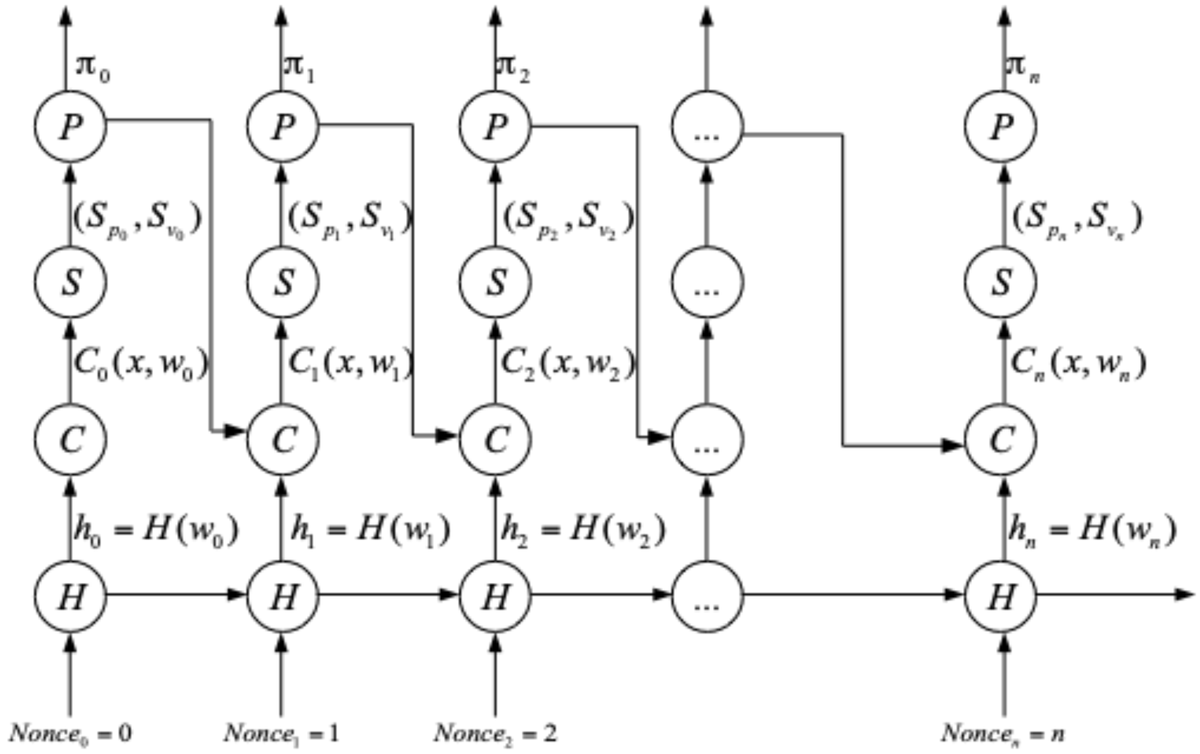


Fig. 4 — Scheme of forming a chain of recursive proofs of computational integrity

Our solutions and results. Test case #2

For a test case, consider a simplified version from test case #1, supplemented by digital signature verification. Each block contains three fields:

- Unique block number: $Nonce_i$;
- Hash of the previous block: h_{i-1};
- Digital signature: EDS_i.

Additionally, each hash h_i is encrypted with a secret key *sk*, i.e. we form a signature $EDS_i = EDS(h_i, sk)$.

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.

Our solutions and results. Test case #2: Aggregation

Condition fulfillment $V(S_{Vi}, X_i, \prod_i) = accept$ means that the proof verification $\prod_{i-1} = P(SP_{i-1}, X_{i-1}, W_{i-1})$, $\pi H_i = P(SH_{pi}, xH_i, wH_i)$ 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:

- 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;
- There is a proof of CI of current hash, i.e. the value $h_i = H(h_{i-1}||i)$ is computed correctly;
- There is a proof of CI of current signature, i.e. verification $h_i = D(EDS_i, pk)$ is computed correctly.

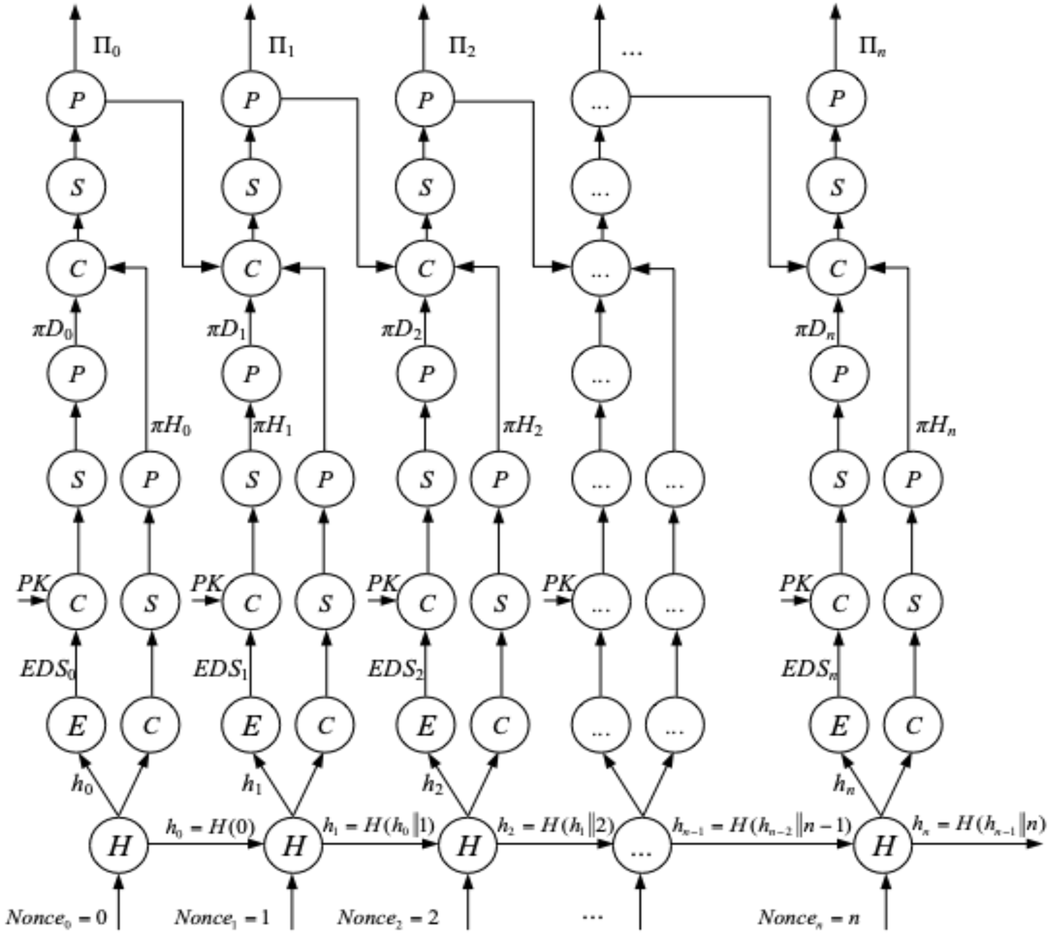


Fig. 5 — Scheme of forming a chain of recursive proofs of CI with digital signature verification

Our solutions and results.

Test case #2 (Aggregation) (1,8 GHz Intel Core i5)

Table 1 — Time and measurement results for a chain of proofs for hashes

№	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

Our solutions and results.

Test case #2 (Aggregation) (1,8 GHz Intel Core i5)

Table 2 — Time and measurement results for a chain of proofs for signatures

№	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

Our solutions and results.

Test case #2 (Aggregation) (1,8 GHz Intel Core i5)

Table 3 — Time and measurement results for a recursive chain of proofs for hashes & signatures

№	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

Our solutions and results.

Test case #2 (Aggregation) (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz)

Table 4 — Time and measurement results for a chain of proofs for hashes

№	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

Our solutions and results.

Test case #2 (Aggregation) (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz)

Table 5 — Time and measurement results for a chain of proofs for signatures

№	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

Our solutions and results.

Test case #2 (Aggregation) (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz)

Table 6 — Time and measurement results for a recursive chain of proofs for hashes & signatures

№	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

Our solutions and results.

Test case #2 (Aggregation) (AMD Ryzen 7 5800U (16) @ 1.900GHz)

Table 7 — Time and measurement results for a chain of proofs for hashes

№	Hash	Time to build a circuit, s	Time to make a proof, s	Proof size, bytes	Verification, s
0	5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9	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

Our solutions and results.

Test case #2 (Aggregation) (AMD Ryzen 7 5800U (16) @ 1.900GHz)

Table 8 — Time and measurement results for a chain of proofs for signatures

№	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

Our solutions and results.

Test case #2 (Aggregation) (AMD Ryzen 7 5800U (16) @ 1.900GHz)

Table 9 — Time and measurement results for a chain of proofs for signatures

№	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

Our solutions and results.

Test case #2 (Recursion) (AMD Ryzen 7 5800U (16) @ 1.900GHz)

Table 10 — Time and measurement results for a recursive chain of proofs for hashes & signatures

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCDD3FDD5E84F0CAD62B08F2D10B57F9A831C13451E5C5C80A5	0.5414246	0.9578	0.0063	149084
1	AC8BE15C3CC494661BC32FF3457E273A98896338195A812702108574D9930EAD	29.565964	78.2291	0.0174	211432
2	3730496E35571584A2839C243481A461AA7E304203666CB6A358247CB2818914	29.53088	59.7240	0.0192	211432
3	C9C25FCF5183B139A462ECD06919572A88059355D3271F5CFDBBCD46F32C6723	30.960339	74.2933	0.0382	211432
4	81DDC8D248B2DCCDD3FDD5E84F0CAD62B08F2D10B57F9A831C13451E5C5C80A5	32.037373	84.5603	0.0279	211432

Our solutions and results.

Test case #2 (Recursion) (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz)

Table 11 — Time and measurement results for a recursive chain of proofs for hashes & signatures

№	Hash	Time to build a circuit, s	Time to prove, s	Verification, s	Proof size, bytes
0	81DDC8D248B2DCCDD3FDD5E84F0CAD62B08F2D10B57F9A831C13451E5C5C80A5	1.119945	1.4125	0.0213	149084
1	AC8BE15C3CC494661BC32FF3457E273A98896338195A812702108574D9930EAD	72.49396	74.2432	0.0457	211432
2	3730496E35571584A2839C243481A461AA7E304203666CB6A358247CB2818914	53.94758	46.4669	0.0337	211432
3	C9C25FCF5183B139A462ECD06919572A88059355D3271F5CFDBBCD46F32C6723	50.751015	48.2629	0.0339	211432
4	81DDC8D248B2DCCDD3FDD5E84F0CAD62B08F2D10B57F9A831C13451E5C5C80A5	52.462822	44.4447	0.0379	211432

Our solutions and results. Test case #3: Current task

As the test case #2, each block contains three fields:

- Unique block number: $Nonce_i$;
- Hash of the previous block: h_{i-1};
- Digital signature: EDS_i.

Additionally, following 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 for the i-th block, which includes h_{i-1}, EDS_i^j signatures are the proof of the connection of two blocks. All EDS_i^j are stored in the header of the $(i+1)$-th block (this is not shown in the figure), and they are used to confirm the i-st 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 1, i.e. coalition confirmation of the i-th block requires EDS_{i+1}^j signatures of at least 2/3 of producers.

Our solutions and results. Test case #3: Current task

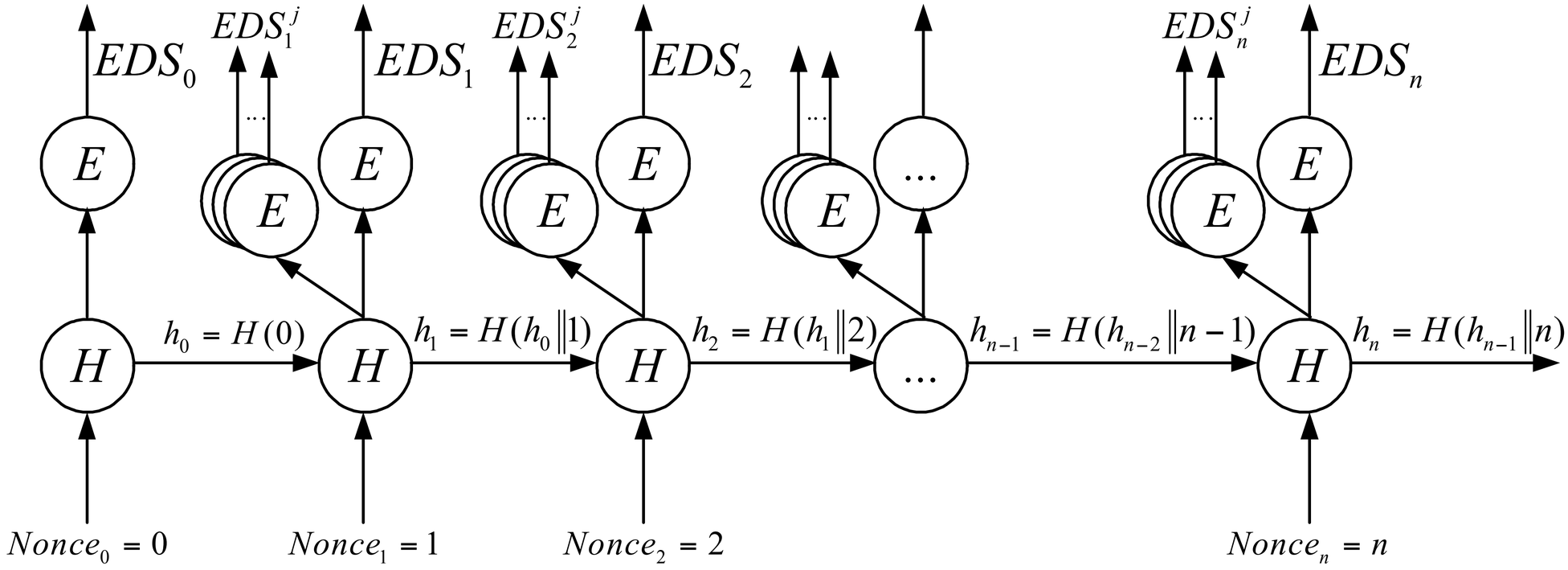


Fig. 6 — A simplified version of the linked list with the formation of digital signatures of all producers

Our solutions and results. Test case #3: Current task

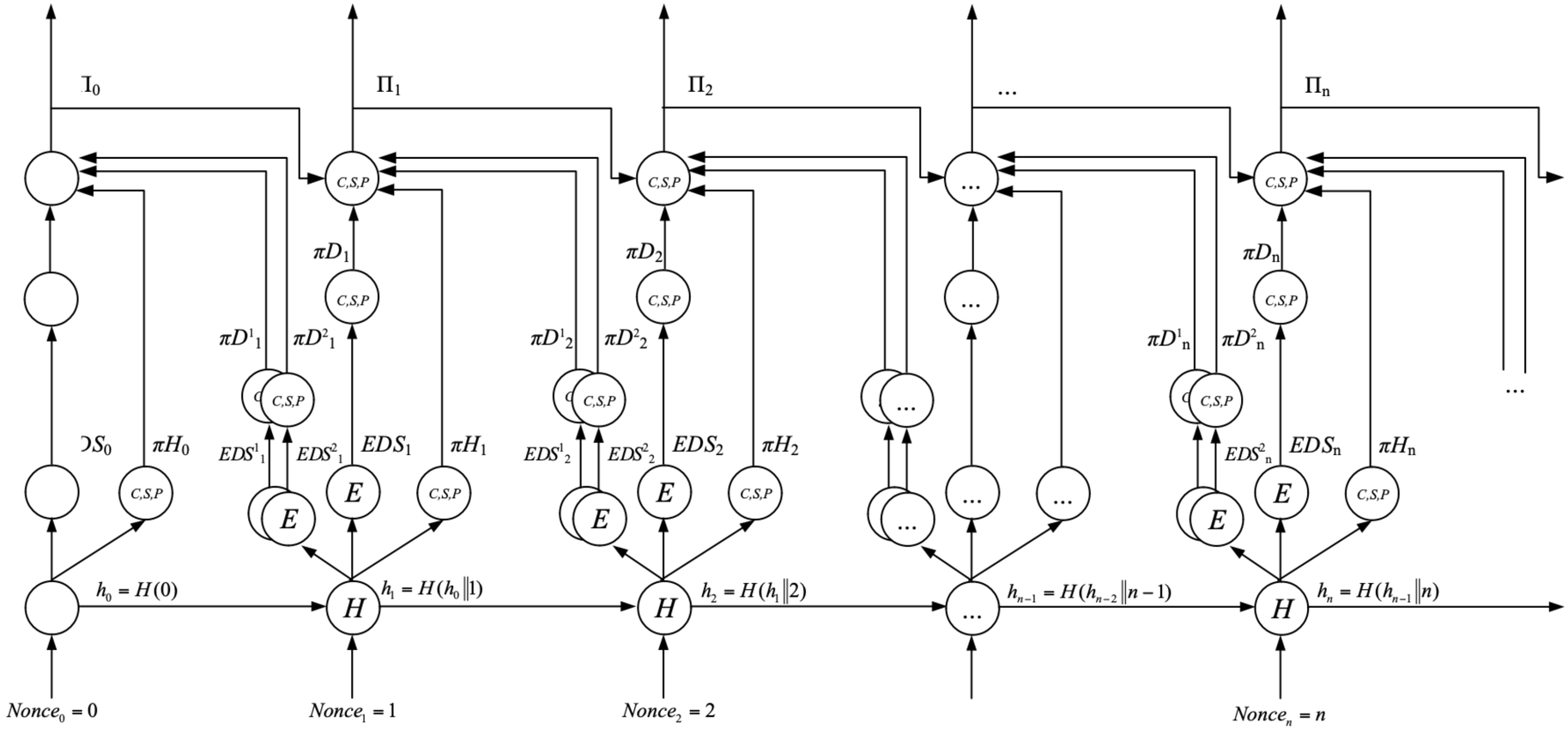


Fig. 7 — Scheme for generating a chain of recursive proofs of CI with digital signature verification of all producers

Conclusions

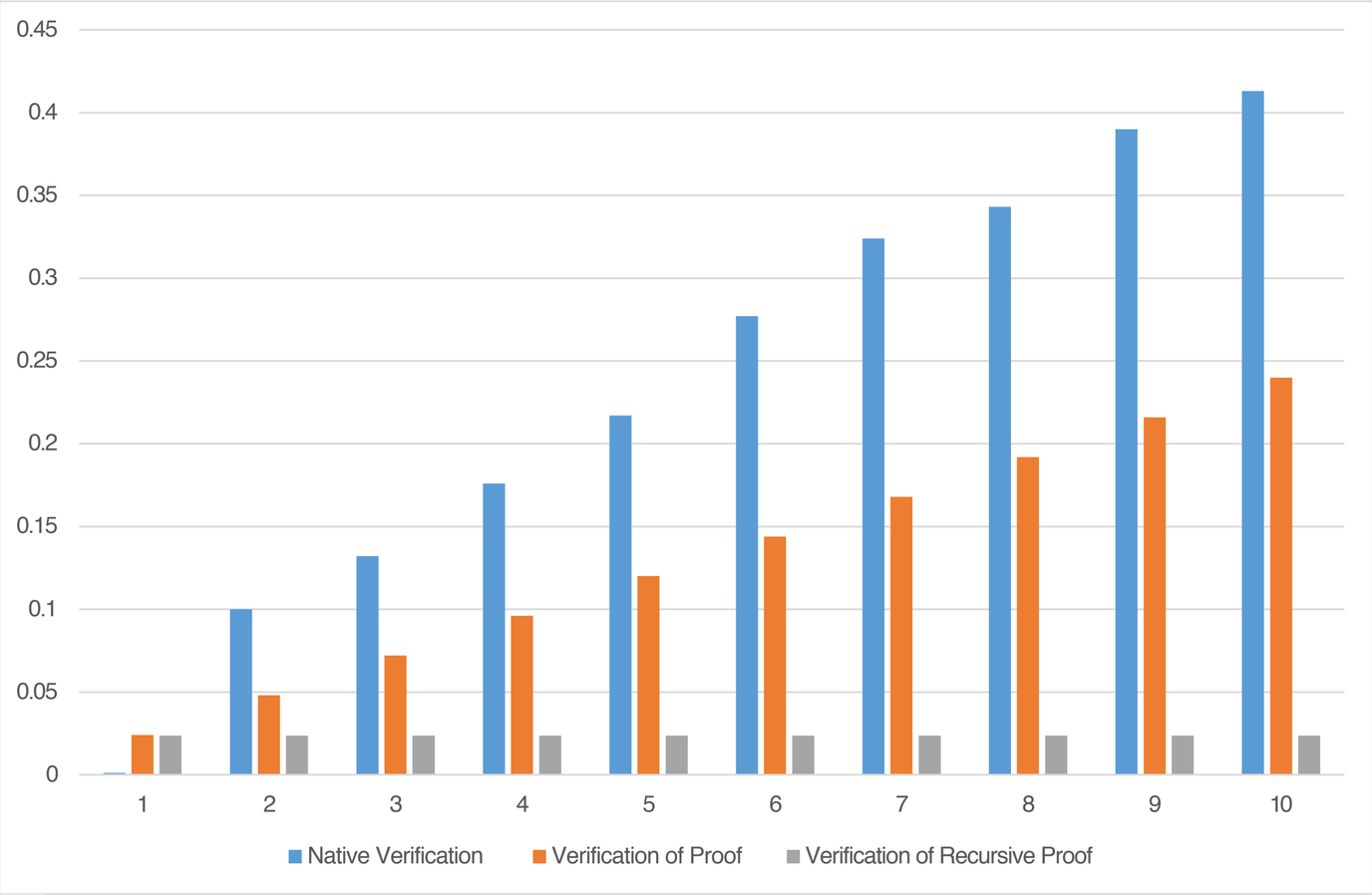


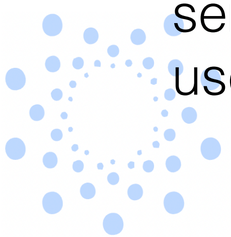
Fig. 8 — Computational complexity of native verification, proof verification, recursive proof verification

Conclusions

Zero-knowledge proofs are a great solution to solve the problems of scalability and confidentiality. This is especially important in large distributed computing projects. We can introduce a proof system and replace native block verification with it. This significantly speeds up verification and facilitates the work of light clients.

Recursive proofs allow building complex schemes of linked proofs. Recursion connects separate proofs into a single chain. A successful verification of a recursion proof for a block means the verification of the chain of the previous blocks. This solution makes the verification procedure easier. We are talking about creating a recursive proof service that allows verifying on the wing any block in the network, guaranteeing its authenticity and security.

We have learned how to build recursive proof systems and our test cases show that this solves the problem of blockchain scalability. Our implementations of separate examples (hashes and digital signatures) and then recursion schemes are a prototype of a full-scale service. We are on our way to build this system and are ready to introduce it to millions of users.



Thank you for the attention!