In-EVM Solana State Verification

Technical Reference

Alisa Cherniaeva
a.cherniaeva@nil.foundation
=nil; Crypto3 (https://crypto3.nil.foundation)

Ilia Shirobokov
i.shirobokov@nil.foundation
=nil; Crypto3 (https://crypto3.nil.foundation)

Mikhail Komarov
nemo@nil.foundation
=nil; Foundation (https://nil.foundation)

December 6, 2021

Contents

Chapter 1

Introduction

This document is a technical reference to the in-EVM Solana's 'Light-Client' state verification project.

1.1 Overview

The project's purpose is to provide Ethereum users with reliable Solana's cluster state and necessary transactions proof.

The project UX consists of several steps:

1. Retrieve Solana's 'Light-Client' state.
2. Generate a proof for it.
3. Submit the proof to EVM-enabled cluster.
4. Verify the proof with EVM.

Such a UX defines projects parts:

1. Solana's 'Light-Client' state retriever.
2. State proof generator.
3. Ethereum RPC proof submitter.
4. EVM-based proof verificator.

Each of these parts will be considered independently.

Chapter 2

State Proof Generator

This introduces a description for Solana's 'Light-Client' state proof generator.
Crucial components which define this part design and performance are:

1. Input data format ('Light-Client' state data structure).
2. Proof system used for the proof generation.
3. Circuit definition used for the proof system.

2.1 'Light-Client' State

Block Information $\bar{B}_k$ is defined as follows:

- k - the number of the block
- $B_k = H(B_{k-1}||\texttt{account\_hash}||\texttt{signature\_count\_buf}||b_k||\texttt{validators\_state})$ - bank hash of the block[1]
- b_k Merkle Block
- B_{k-1} - the previous block's bank hash
- `validators_state` is not implemented for now.

Proof algorithm input is defined as follows:

- n_1 - current confirmed block number
- n_2 - new confirmed block number
- $\{\bar{B}_{n_1}, \ldots, \bar{B}_{n_2}, \ldots, \bar{B}_{n_2+32}\}$ - block information for blocks from n_1 to $n_2 + 32$.
- $\sigma_0, \ldots, \sigma_N$ - signatures for B_{n_2+32}

Approximate code representation of such a state data structure is as follows:

```
template<typename Hash>
struct block_data {
    typedef typename Hash::digest_type digest_type;

    std::size_t block_number;
    digest_type bank_hash;
    digest_type merkle_hash;
    digest_type previous_bank_hash;
    //    std::vector<vote_state> votes;
};

template<typename Hash, typename SignatureSchemeType>
struct state_type {
    typedef Hash hash_type;
    typedef SignatureSchemeType signature_scheme_type;
    typedef typename signature_scheme_type::signature_type signature_type;
```

[1] See https://docs.solana.com/proposals/simple-payment-and-state-verification#block-headers

```
    std::size_t n_1 confirmed;
    std::size_t n_2 new_confirmed;
    std::vector<block_data<hash_type>> repl_data;
    std::vector<signature_type> signatures;
};
```

Validator state-representing data structure (`vote_state`) supposes such a state to begin being handled by Solana replication protocol (or its implementation) for handling the tracking of votes state being unchanged 'till the end of epoch.

2.2 Proof System

WIP

The proof system used for proving Solana's 'Light-Client' state on EVM is Redshift SNARK[1]. RedShift is a transparent SNARK that uses PLONK[2] proof system but replaces the commitment scheme. Initial paper proposal is to employ FRI[3] protocol to obtain transparency for the PLONK system.

However, FRI cannot be straightforwardly used with the PLONK system. To achieve the required security level without huge overheads, the authors introduce *list polynomial commitment* scheme as a part of the protocol. For more details, the reader gets referred to [1].

The original RedShift protocol utilizes the classic PLONK[2] system. To provide better performance, the original protocol is generalized to be used with PLONK with custom gates [4], [5] and lookup arguments [6], [7].

2.3 Optimizations

WIP

2.3.1 Batched FRI

Instead of check each commitment individualy, we can aggregate them for FRI. For polynomials $f_0, \ldots, f_k$:

1. Get θ from transcript
2. $f = f_0 \cdot \theta^{k-1} + \cdots + f_k$
3. Run FRI over f, using oracles to $f_0, \ldots, f_k$

Thus, we can run only one FRI instance for all commited polynomials.
See [1] for details.

2.3.2 Hash By Column

Instead of committing each of the polynomials, we can use the same Merkle tree for several polynomials. It decreases the number of Merkle tree paths that need to be provided by the prover.
See [8], [1] for details.

2.3.3 Hash By Subset

On the each $i+1$ FRI round, the prover should send all elements from a coset $H \in D^{(i)}$. Each Merkle leaf is able to contain the whole coset instead of separate values.
See [8] for details. Similar approach is described in [1]. However, the authors of [1] use more values per leaf, that leads to better performance.

2.4 RedShift Protocol

WIP
Notations:

$N_{\texttt{wires}}$	Number of wires ('advice columns')
$N_{\texttt{perm}}$	Number of wires that are included in the permutation argument
$N_{\texttt{sel}}$	Number of selectors used in the circuit
$N_{\texttt{const}}$	Number of constant columns
$N_{\texttt{lookups}}$	Number of lookups
$\mathbf{f}_i$	Witness polynomials, $0 \leq i < N_{\texttt{wires}}$
$\mathbf{f}_{c_i}$	Constant-related polynomials, $0 \leq i < N_{\texttt{const}}$
$\mathbf{gate}_i$	Gate polynomials, $0 \leq i < N_{\texttt{sel}}$
$\sigma(\text{col} : i, \text{row} : j) = (\text{col} : i', \text{row} : j')$	Permutation over the table

For details on polynomial commitment scheme and polynomial evaluation scheme, we refer the reader to [1].

---

1. $\mathcal{L}' = (\mathbf{q}_0, ..., \mathbf{q}_{N_{\texttt{sel}}})$
2. Let ω be a 2^k root of unity
3. Let δ be a T root of unity, where $T \cdot 2^S + 1 = p$ with T odd and $k \leq S$
4. Compute $N_{\texttt{perm}}$ permutation polynomials $S_{\sigma_i}(X)$ such that $S_{\sigma_i}(\omega^j) = \delta^{i'} \cdot \omega^{j'}$
5. Compute $N_{\texttt{perm}}$ identity permutation polynomials: $S_{id_i}(X)$ such that $S_{id_i}(\omega^j) = \delta^i \cdot \omega^j$
6. Let $H = \{\omega^0, ..., \omega^n\}$ be a cyclic subgroup of $\mathbb{F}^*$
7. Let $Z(X) = \prod a \in H^*(X - a)$
8. Let A_i be a witness lookup columns and S_i be a table columns, $i = 0, .., m$.

Preprocessing:

2.4.1 Prover View

1. Choose masking polynomials:

$$h_i(X) \leftarrow \mathbb{F}_{<k}[X] \text{ for } 0 \leq i < N_{\texttt{wires}}$$

   Remark: For details on choice of k, we refer the reader to [1].

2. Define new witness polynomials:

$$f_i(X) = \mathbf{f}_i(X) + h_i(X)Z(X) \text{ for } 0 \leq i < N_{\texttt{wires}}$$

3. Add commitments to f_i to transcript
4. Get $\theta \in \mathbb{F}$ from $hash$(transcript)
5. Construct the witness lookup compression and table compression $S(\theta)$ and $A(\theta)$:

$$A(\theta) = \theta^{m-1}A_0 + \theta^{m-2}A_1 + ... + \theta A_{m-2} + A_{m-1}$$
$$S(\theta) = \theta^{m-1}S_0 + \theta^{m-2}S_1 + ... + \theta S_{m-2} + S_{m-1}$$

6. Produce the permutation polynomials $S'(X)$ and $A'(X)$ such that:

6.1 All the cells of column A' are arranged so that like-valued cells are vertically adjacent to each other.

6.2 The first row in a sequence of values in A' is the row that has the corresponding value in S'.

7. Compute and add commitments to A' and S' to transcript

8. Get $\beta, \gamma \in \mathbb{F}$ from $hash$(transcript)

9. For $0 \leq i < N_{\texttt{perm}}$

$$p_i = f_i + \beta \cdot S_{id_i} + \gamma$$
$$q_i = f_i + \beta \cdot S_{\sigma_i} + \gamma$$

10. Define:

$$p'(X) = \prod_{0 \leq i < N_{\texttt{perm}}} p_i(X) \in \mathbb{F}_{<N_{\texttt{perm}} \cdot n}[X]$$
$$q'(X) = \prod_{0 \leq i < N_{\texttt{perm}}} q_i(X) \in \mathbb{F}_{<N_{\texttt{perm}} \cdot n}[X]$$

11. Compute $P(X), Q(X) \in \mathbb{F}_{<n+1}[X]$, such that:

$$P(\omega) = Q(\omega) = 1$$
$$P(\omega^i) = \prod_{1 \leq j < i} p'(\omega^i) \text{ for } i \in 2, \ldots, n+1$$
$$Q(\omega^i) = \prod_{1 \leq j < i} q'(\omega^i) \text{ for } i \in 2, \ldots, n+1$$

12. Compute and add commitments to P and Q to transcript

13. Compute permutation product column:

$$V(\omega^i) = \frac{(\theta^{m-1}A_0(\omega^i) + \theta^{m-2}A_1(\omega^i) + \ldots + \theta A_{m-2}(\omega^i) + A_{m-1}(\omega^i) + \beta) \cdot (\theta^{m-1}S_0(\omega^i) + \theta^{m-2}S_1(\omega^i) + \ldots + \theta S_{m-2}(\omega^i) + S_{m-1}(\omega^i) + \gamma)}{(A'(\omega^i) + \beta)(S'(\omega^i) + \gamma)}$$
$$V(1) = V(\omega^{N_{\texttt{lookups}}}) = 1$$

14. Compute and add commitments to V to transcript

15. Get $\alpha_0, \ldots, \alpha_5 \in \mathbb{F}$ from $hash$(transcript)

16. Get τ from $hash$(transcript)

17. Define polynomials ($F_0, \ldots, F_4$ - copy-satisfability, $\texttt{gate}_0$ is PI-constraining gate):

$$F_0(X) = L_1(X)(P(X) - 1)$$
$$F_1(X) = L_1(X)(Q(X) - 1)$$
$$F_2(X) = P(X)p'(X) - P(X\omega)$$
$$F_3(X) = Q(X)q'(X) - Q(X\omega)$$
$$F_4(X) = L_n(X)(P(X\omega) - Q(X\omega))$$
$$F_5(X) = \sum_{0 \leq i < N_{\texttt{sel}}} (\tau^i \cdot \mathbf{q}_i(X) \cdot \texttt{gate}_i(X)) + PI(X)$$

18. For the lookup:

18.1 Two selectors q_{last} and q_{blind} are used, where $q_{last} = 1$ for t last blinding rows and $q_{blind} = 1$ on the row in between the usable rows and the blinding rows.

18.2 $F_6(X) = L_0(X)(1 - V(X))$

18.3 $F_7(X) = q_{last} \cdot (V(X)^2 - V(X))$

18.4 $F_8(X) = (1 - (q_{last} + q_{blind})) \cdot (V(\omega X)(A'(X) + \beta)(S'(X) + \gamma) - V(X)(\theta^{m-1}A_0(X) + ... + A_{m-1}(X) + \beta)(\theta^{m-1}S_0(X) + ... + S_{m-1}(X) + \gamma))$

18.5 $F_9(X) = L_0(X) \cdot (A'(X) - S'(X))$

18.6 $F_{10}(X) = (1 - (q_{last} + q_{blind})) \cdot (A'(X) - S'(X)) \cdot (A'(X) - A'(\omega^{-1}X))$

19. Compute:

$$F(X) = \sum_{i=0}^{10} \alpha_i F_i(X)$$
$$T(X) = \frac{F(X)}{Z(X)}$$

20. $N_T \coloneqq \mathtt{max}(N_{\mathtt{perm}}, \mathtt{deg}_{\mathtt{gates}} - 1)$, where $\mathtt{deg}_{\mathtt{gates}}$ is the highest degree of the degrees of gate polynomials.

21. Split $T(X)$ into separate polynomials $T_0(X), ..., T_{N_T-1}(X)$[2]

22. Add commitments to $T_0(X), ..., T_{N_T-1}(X)$ to transcript

23. Get $y \in \mathbb{F}/H$ from $hash|_{\mathbb{F}/H}$(transcript)

24. Run evaluation scheme with the committed polynomials and y
Remark: Depending on the circuit, evaluation can be done also on $y\omega, y\omega^{-1}$.

25. The proof is $\pi_{\mathtt{comm}}$ and $\pi_{\mathtt{eval}}$„ where:

- $\pi_{\mathtt{comm}} = \{f_{0,\mathtt{comm}}, \ldots, f_{N_{\mathtt{wires}}-1,\mathtt{comm}}, P_{\mathtt{comm}}, Q_{\mathtt{comm}}, T_{0,\mathtt{comm}}, ..., T_{N_T-1,\mathtt{comm}}, A'_{\mathtt{comm}}, S'_{\mathtt{comm}}, V_{\mathtt{comm}}\}$
- $\pi_{\mathtt{eval}}$ is evaluation proofs for $f_0(y), \ldots, f_{N_{\mathtt{wires}}}(y), P(y), P(y\omega), Q(y), Q(y\omega)$, $T_0(y), \ldots, T_{N_T-1}(y), A'(y), A'(y\omega^{-1}), S'(y), V(y), V(y\omega)$

2.4.2 Verifier View

1. Let $f_{0,\mathtt{comm}}, \ldots, f_{N_{\mathtt{wires}}-1,\mathtt{comm}}$ be commitments to $f_0(X), \ldots, f_{N_{\mathtt{wires}}-1}(X)$
2. transcript = setup__values$||f_{0,\mathtt{comm}}|| \ldots ||f_{N_{\mathtt{wires}}-1,\mathtt{comm}}$
3. $\theta = hash$(transcript)
4. Let $A'_{\mathtt{comm}}, S'_{\mathtt{comm}}$ be commitments to $A'(X), S'(X)$.
5. transcript = transcript$||A'_{\mathtt{comm}}||S'_{\mathtt{comm}}$
6. $\beta, \gamma = hash$(transcript)
7. Let $P_{\mathtt{comm}}, Q_{\mathtt{comm}}, V_{i,\mathtt{comm}}$ be commitments to $P(X), Q(X), V(X)$.
8. transcript = transcript$||P_{\mathtt{comm}}||Q_{\mathtt{comm}}||V_{\mathtt{comm}}$
9. $\alpha_0, \ldots, \alpha_5 = hash$(transcript)
10. $\tau = hash$(transcript)
11. $N_T \coloneqq \mathtt{max}(N_{\mathtt{perm}}, \mathtt{deg}_{\mathtt{gates}} - 1)$, where $\mathtt{deg}_{\mathtt{gates}}$ is the highest degree of the degrees of gate polynomials.
12. Let $T_{0,\mathtt{comm}}, ..., T_{N_T-1,\mathtt{comm}}$ be commitments to $T_0(X), ..., T_{N_T-1}(X)$
13. transcript = transcript$||T_{0,\mathtt{comm}}||...||T_{N_T-1,\mathtt{comm}}$
14. $y = hash_{\mathbb{F}/H}$(transcript)
15. Run evaluation scheme verification with the committed polynomials and y to get values $f_i(y), P(y), P(y\omega), Q(y), Q(y\omega), T_j(y), A'(y), S'(y), V(y), A'(y\omega^{-1}), V(y\omega)$.
Remark: Depending on the circuit, evaluation can be done also on $f_i(y\omega), f_i(y\omega^{-1})$ for some i.
16. Calculate:

$$F_0(y) = L_1(y)(P(y) - 1)$$
$$F_1(y) = L_1(y)(Q(y) - 1)$$
$$p'(y) = \prod p_i(y) = \prod f_i(y) + \beta \cdot S_{id_i}(y) + \gamma$$
$$F_2(y) = P(y)p'(y) - P(y\omega)$$
$$q'(y) = \prod q_i(y) = \prod f_i(y) + \beta \cdot S_{\sigma_i}(y) + \gamma$$
$$F_3(y) = Q(y)q'(y) - Q(y\omega)$$

[2]Commit scheme supposes that polynomials should be degree $\leq n$

$$F_4(y) = L_n(y)(P(y\omega) - Q(y\omega))$$
$$F_5(y) = \sum_{0 \le i < N_{\text{sel}}} (\tau^i \cdot \mathbf{q}_i(y) \cdot \texttt{gate}_i(y)) + PI(y)$$
$$T(y) = \sum_{0 \le j < N_T} y^{n \cdot j} T_j(y) \;\; F_6(y) = L_0(y)(1 - V(y))$$
$$F_7(y) = q_{last} \cdot (V(y)^2 - V(y))$$
$$F_8(y) = (1 - (q_{last} + q_{blind})) \cdot (V(\omega y)(A'(y) + \beta)(S'(y) + \gamma) - V(y)(\theta^{m-1} A_0(y) + ... + A_{m-1}(y) + \beta)(\theta^{m-1} S_{i,0}(y) + ... + S_{m-1}(y) + \gamma))$$
$$F_9(y) = L_0(y) \cdot (A'(y) - S'(y))$$
$$F_{10}(y) = (1 - (q_{last} + q_{blind})) \cdot (A'(y) - S'(y)) \cdot (A'(y) - A'(\omega^{-1} y))$$

17. Check the identity:

$$\sum_{i=0}^{10} \alpha_i F_i(y) = Z(y) T(y)$$

2.5 Circuit Definition

This section contains a description of PLONK-style circuits for In-EVM Solana's "Light Client" state verification[3].

This section provides a high-level overview of the circuit used for proof generation and verification. Following sections provide sub-circuits details.

2.5.1 Verification Circuit Overview

Let bank-hashes of proving block set be $\{H_{B_{n_1}}, ..., H_{B_{n_2}}\}$. The last confirmed block is H_{B_L}. Each positively confirmed block is signed by M validators.

Denote by `block_data` the data that is included in the bank hash other than the bank hash of the parent block.

1. $H_{B_{n_1}} = H_{B_L}$ // H_{B_L} is a public input
2. Validator set constraints. // see Section 2.5.9
3. for i from $n_1 + 1$ to $n_2 + 32$:
   3.1 $H_{B_i} = \texttt{sha256}(\texttt{block\_data} || H_{B_{i-1}})$ // see Section 2.5.2
4. for j from 0 to M:
   4.1 Ed25519 constraints for $H_{B_{n_2+32}}$ // see Section 2.5.6
5. Merkle tree constraints for the set $\{H_{B_{n_1}}, ..., H_{B_{n_2}}\}$ // see Section 2.5.5

2.5.2 SHA-256 Circuit

Suppose that input data is in the 32-bits form, which is already padded to the required size. We suppose that the checking that chunked input data corresponds to the original data out of the circuit. However, we do not need to range constrain these chunks as we get them for free from the SHA-256 circuit.

Thus, the preprocessing constraints for the SHA-256 circuit is a decomposition of k message blocks to 32 bits chunks without range proofs. For 'Solana-EVM' circuit, $k = 3$.

Lookup tables We use the following lookup tables:

1. **SHA-256 NORMALIZE4** with 2 columns and 2^{14} rows. The first column contains all possible 14-bits words. The second column contains corresponding sparse representations with base 4. The constraints can be used for the range check and sparse representation simultaneously.
2. **SHA-256 NORMALIZE7** with 2 columns and 2^{14} rows. The first column contains all possible 14-bits words. The second column contains corresponding sparse representations with base 7. The constraints can be used for the range check and sparse representation simultaneously.

[3] https://blog.nil.foundation/2021/10/14/solana-ethereum-bridge.html

3. **SHA-256 NORMALIZE MAJ** with 2 columns and 2^8 rows. The first column contains all possible 8-bits words. The second column contains corresponding sparse representations with base 4.

4. **SHA-256 NORMALIZE CH** with 2 columns and 2^8 rows. The first column contains all possible 8-bits words. The second column contains corresponding sparse representations with base 7.

Message scheduling For each block of 512 bits of the padded message the 64 words are constructed in the following way:

- The first 16 words are obtained by splitting the message.
- The last 48 words are obtained by using the functions σ_0, σ_1:

$$W_i = \sigma_1(W_{i-2}) \oplus W_{i-7} \oplus \sigma_0(W_{i-15}) \oplus W_{i-16}$$

Each round of the message scheduling has the following table:

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	a	a_0	a_1	a_2	a_3	$\hat{a}_1$	$\hat{a}_2$	a'_0	
j + 1	W_i	W_j	a'_1	a'_2	a'_3	s_0	s_1	s_2	s_3
j + 2	w	b'_0	b'_1	b'_2	b'_3	s_0	s_1	s_2	s_3
j + 3	b	b_0	b_1	b_2	b_3	$\hat{b}_0$	$\hat{b}_1$	$\hat{b}_3$	

The first 16 words require a range check. We get it fo free from range-constraining chunks inside functions σ_0 and σ_1. Thus, for i from 16 to 63:

1. Apply σ_0 to W_{i-15}.

2. Add the following constraint for W_i:

$$w_{1,j+2} = w_{1,j+1} + w_{2,j+1} + w_{6,j+1} + w_{7,j+1} \cdot 2^3 + w_{8,j+1} \cdot 2^7 + w_{9,j+1} \cdot 2^{18} + w_{6,j+2} + w_{7,j+2} \cdot 2^{10} + w_{8,j+2} \cdot 2^{17} + w_{9,j+2} \cdot 2^{19},$$

3. Apply σ_1 to W_{i-2}.

Thus, the message schedule takes $4 \cdot 48 = 192$ rows.

The function σ_0 contains sparse mapping with base 4. Let a be divided to chunks a_0, a_1, a_2, a_3 which equals to $3, 4, 11, 14$ bits respectively. The values a'_0, a'_1, a'_2, a'_3 are in sparse form, and a' is a sparse a. **SHA-256 NORMALIZE4** lookup table is used for mapping to sparse representation and range-constraining for each chunk a_i, where bit-length of $a_i > 3$. If a chunk is 14 bits long, then it is constrained for free. Else the prover has to calculate the sparse representation $\hat{a}_i$ for $2^j \cdot a_i$, where $j + \texttt{len}(a_i) = 14$ and $\texttt{len}(a_i)$ is bit-length of a_i.

Constraints:

$$w_{1,j+0} = w_{2,j+0} + w_{3,j+0} \cdot 2^3 + w_{4,j+0} \cdot 2^7 + w_{5,j+0} \cdot 2^{18}$$
$$(w_{2,j+0} - 7) \cdot (w_{2,j+0} - 6) \cdot ... \cdot w_{2,j+0} = 0$$
10 plookup constraints: $(w_{2,j+0}, w_{8,j+0}), (2^{10} \cdot w_{3,j+0}, w_{6,j+0}), (w_{3,j+0}, w_{3,j+1}), (2^3 \cdot w_{4,j+0}, w_{7,j+0}), (w_{4,j+0}, w_{4,j+1}), (w_{5,j+0}, w_{5,j+1}), (w_{6,j+1}, (w_{4,j+1} + w_{5,j+1} + w_{3,j+1}), (w_{7,j+1}, (w_{5,j+1} + w_{8,j+0} + w_{4,j+1}, (w_{8,j+1}, (w_{8,j+0} + w_{3,j+1} + w_{5,j+1}), (w_{9,j+1}, (w_{3,j+1} + w_{4,j+1})$

The function σ_1 contains sparse mapping subcircuit with base 4. Let a be divided to chunks a_0, a_1, a_2, a_3 which equals to $10, 7, 2, 13$ bits respectively. The values a'_0, a'_1, a'_2, a'_3 are in sparse form and a' is a sparse a. **SHA-256 NORMALIZE4** lookup table is used for mapping to sparse representation and range-constraining in the same way as for σ_0.

Constraints:

$$w_{1,j+3} = w_{2,j+3} + w_{3,j+3} \cdot 2^{10} + w_{4,j+3} \cdot 2^{17} + w_{5,j+3} \cdot 2^{19}$$
$$(w_{4,j+3} - 3) \cdot (w_{4,j+3} - 2) \cdot (w_{4,j+3} - 1) \cdot w_{4,j+3} = 0$$
11 plookup constraints: $(2^4 \cdot (w_{2,j+3}, w_{6,j+3}), (2^7 \cdot w_{3,j+3}, w_{7,j+3}), (2 \cdot w_{5,j+3}, w_{8,j+3}), (w_{2,j+3}, w_{2,j+2}), (w_{3,j+3}, w_{3,j+2}), (w_{4,j+3}, w_{4,j+2}), (w_{5,j+3}, w_{5,j+2}), (w_{6,j+2}, (w_{4,j+2} + w_{5,j+2} + w_{3,j+2})), (w_{7,j+2}, (w_{5,j+2} + w_{2,j+2} + w_{4,j+2}, (w_{8,j+2}, (w_{2,j+2} + w_{3,j+2} + w_{5,j+2}), (w_{9,j+2}, (w_{3,j+2} + w_{4,j+2})$

Compression There are 64 rounds of compression. Each round of compression has the following table:

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	e	e'_0	e_0	e_1	e_2	e_3	$\hat{e}_1$	$\hat{e}_2$	$\hat{e}_3$
j + 1	e'	f'	e'_1	e'_2	e'_3	s_0	s_1	s_2	s_3
j + 2	$ch_{0,sparse}$	$ch_{1,sparse}$	$ch_{2,sparse}$	$ch_{3,sparse}$	e_{new}	ch_0	ch_1	ch_2	ch_3
j + 3	g'	d	h	W_r	a_{new}	maj_3	maj_0	maj_1	maj_2
j + 4	$maj_{0,sparse}$	$maj_{1,sparse}$	$maj_{2,sparse}$	$maj_{3,sparse}$	c'	s_0	s_1	s_2	s_3
j + 5	a'	b'	a'_0	a'_1	a'_2	a'_3			
j + 6	a		a_0	a_1	a_2	a_3	$\hat{a}_0$	$\hat{a}_1$	$\hat{a}_3$

The working variables a, b, c, d, e, f, g, h equals to the fixed initial $SHA-256$ values for the first chunk and to the sum of previous output and initial values for the rest of chunks. The variables with quotes are corresponded sparse representation. For each chunk, the following rows are used:

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	a	a'	b	b'	d	–	–	–	–
j + 1	c	c'	e	e'	h	–	–	–	–
j + 2	f	f'	g	g'	–	–	–	–	–

For the first round, $a, a', b', c', d, e, e', f', g', h$ are copy constrained with corresponded values from the table above.

For the second round, b', c', d, f', g', h are copy constrained with a', b', c, e', f', g from the table. The values a, e are copy constrained with a_{new}, e_{new} from the previous round.

For the third round, c', d, g', h are copy constrained with a', b, e', f. The values a, e are copy constrained with a_{new}, e_{new} from the previous round. The values b', f' are copy constrained with a', e' from the previous round.

In the rest of the rounds the following 'non-special' copy constraints are used:

1. The values a, e are copy constrained with a_{new}, e_{new} from the previous round.
2. The values b', f' are copy constrained with a', e' from the previous round.
3. The values c', g' are copy constrained with b', c' from the previous round.
4. The values d, h are copy constrained with a', e' from the round $r-3$, where r is current round.

The Σ_0 function contains subcircuit with base 4. Let a be divided to chunks a_0, a_1, a_2, a_3 which equals to $2, 11, 9, 10$ bits respectively. The values a'_0, a'_1, a'_2, a'_3 are in sparse form and a' is a sparse a. **SHA-256 NORMALIZE4** lookup table is used for mapping to sparse representation and range-constraining in the same way as for σ_0.

Constraints:

$$w_{1,j+6} = w_{3,j+6} + w_{4,j+6} \cdot 2^2 + w_{5,j+6} \cdot 2^{13} + w_{6,j+6} \cdot 2^{22}$$
$$w_{1,j+5} = w_{3,j+6} + w_{4,j+6} \cdot 4^2 + w_{5,j+6} \cdot 4^{13} + w_{6,j+6} \cdot 4^{22}$$
$$(w_{3,j+6} - 3) \cdot (w_{3,j+6} - 2) \cdot (w_{3,j+6} - 1) \cdot w_{3,j+6} = 0$$
11 plookup constraints: $(2^3 \cdot (w_{4,j+6}, w_{7,j+6}), (2^5 \cdot w_{5,j+6}, w_{8,j+6}), (2^4 \cdot w_{6,j+6}, w_{9,j+6}), (w_{3,j+6}, w_{3,j+5}), (w_{4,j+6}, w_{4,j+5}), (w_{5,j+6}, w_{5,j+5}), (w_{6,j+6}, w_{6,j+5}), (w_{6,j+4}, (w_{4,j+5} + w_{5,j+5} + w_{6,j+5}), (w_{7,j+4}, (w_{5,j+5} + w_{6,j+5} + w_{3,j+5}, (w_{8,j+4}, (w_{6,j+5} + w_{3,j+5} + w_{4,j+5}), (w_{9,j+4}, (w_{3,j+5} + w_{4,j+5} + w_{5,j+5})$

The Σ_1 function contains subcircuit with base 7. Let a be divided to chunks a_0, a_1, a_2, a_3 which equals to $6, 5, 14, 7$ bits respectively. The values a'_0, a'_1, a'_2, a'_3 are in sparse form, and a' is a sparse a. **SHA-256 NORMALIZE7** lookup table is used for mapping to sparse representation and range-constraining in the same way as for σ_0.

Constraints:

$$w_{1,j+0} = w_{3,j+0} + w_{4,j+0} \cdot 2^6 + w_{5,j+0} \cdot 2^{11} + w_{6,j+0} \cdot 2^{25}$$
$$w_{1,j+1} = w_{2,j+0} + w_{3,j+1} \cdot 7^6 + w_{4,j+1} \cdot 7^{11} + w_{5,j+1} \cdot 7^{25}$$
11 plookup constraints: $(2^8 \cdot (w_{3,j+0}, w_{2,j+0}), (2^9 \cdot w_{4,j+0}, w_{3,j+1}), (2^7 \cdot w_{6,j+0}, w_{5,j+1}), (w_{3,j+0}, w_{2,j+0}), (w_{4,j+0}, w_{3,j+1}), (w_{5,j+0}, w_{4,j+1}), (w_{6,j+0}, w_{5,j+1}), (w_{6,j+1}, (w_{3,j+1} + w_{4,j+1} + w_{5,j+1}), (w_{7,j+1}, (w_{4,j+1} + w_{5,j+1} + w_{2,j+0}, (w_{8,j+1}, (w_{5,j+1} + w_{2,j+0} + w_{3,j+1}), (w_{9,j+1}, (w_{2,j+0} + w_{3,j+1} + w_{4,j+1})$

The Maj function contains subcircuit with base 4 for a, b, c. **SHA-256 NORMALIZE MAJ** lookup table is used for mapping to sparse representation in the same way as for σ_0. The value of the *maj* function is stored in chunks of 8 bits. Constraints:

$$w_{1,j+4} + w_{2,j+4} \cdot 4^8 + w_{3,j+4} \cdot 4^{8\cdot 2} + w_{4,j+4} \cdot 4^{8\cdot 3} = w_{1,j+5} + w_{2,j+5} + w_{5,j+4}$$
$$\text{4 plookup constraints: } (w_{6,j+3}, w_{1,j+4}), (w_{7,j+3}, w_{2,j+4}), (w_{8,j+3}, w_{3,j+4}), (w_{9,j+3}, w_{4,j+4})$$

The Ch function contain sparse mapping subcircuit with base 7 for e, f, g. **SHA-256 NORMALIZE CH** lookup table is used for mapping to sparse representation in the same way as for σ_0. The value of the *ch* function is stored in chunks of 8 bits. Constraints:

$$w_{1,j+2} + w_{2,j+2} \cdot 7^8 + w_{3,j+2} \cdot 7^{8\cdot 2} + w_{4,j+2} \cdot 7^{8\cdot 3} = w_{1,j+1} + 2 \cdot w_{2,j+1} + 3 \cdot w_{1,j+3}$$
$$\text{4 plookup constraints: } (w_{6,j+2}, w_{1,j+2}), (w_{7,j+2}, w_{2,j+2}), (w_{8,j+2}, w_{3,j+2}), (w_{9,j+2}, w_{4,j+2})$$

Update the values a and e Constraints:

$$w_{5,j+2} = w_{2,j+3} + w_{3,j+3} + w_{6,j+1} + w_{7,j+1} \cdot 2^6 + w_{8,j+1} \cdot 2^{11} + w_{9,j+1} \cdot 2^{25} + w_{6,j+2} + w_{7,j+2} \cdot 2^8 + w_{8,j+2} \cdot 2^{8\cdot 2} + w_{9,j+2} \cdot 2^{8\cdot 3} + k[r] + w_{4,j+3}, \text{ where } r \text{ is a number of round.}$$

$$w_{5,j+3} = w_{5,j+2} - w_{2,j+3} + w_{6,j+4} + w_{7,j+4} \cdot 2^2 + w_{8,j+4} \cdot 2^{13} + w_{9,j+4} \cdot 2^{22} + w_{6,j+3} + w_{7,j+3} \cdot 2^8 + w_{8,j+3} \cdot 2^{8\cdot 2} + w_{9,j+3} \cdot 2^{8\cdot 3}$$

Cost The total value of rows is $48 \cdot 4 + 7 \cdot 64 + 3 = 643$ per chunk.

2.5.3 SHA2-512 Circuit

SHA-512 uses the similar logical functions as in 2.5.2 which operates on 64-bits words. Thus each input uses the same range proof which extended to 64-bits.

Range proof that $a < 2^{64}$ Let $a = \{a_0, ..., a_{32}\}$, where a_i is two bits.

	w_1	w_2	w_3	w_4	w_o
j + 0	a_{29}	a_{30}	a_{31}	a_{32}	acc
j + 1	a_{25}	a_{26}	a_{27}	a_{28}	acc
...					
j + 6	a_4	a_5	a_6	a_7	acc
j + 7	a_0	a_1	a_2	a_3	a

Range gate constraints:

$$w_{1,i}(w_{1,i}-1)(w_{1,i}-2)(w_{1,i}-3) + w_{2,i}(w_{2,i}-1)(w_{2,i}-2)(w_{2,i}-3) + w_{3,i}(w_{3,i}-1)(w_{3,i}-2)(w_{3,i}-3) + w_{4,i}(w_{4,i}-1)(w_{4,i}-2)(w_{4,i}-3)$$
$$w_{o,i} = w_{o,i-1} \cdot 4^4 + w_{4,i} \cdot 4^3 + w_{3,i} \cdot 4^2 + w_{2,i} \cdot 4 + w_{1,i}$$

The range proofs are included for each input data block.

The function σ_0 contain sparse mapping subcircuit with base 4. Let a be divided to 8 bits-chunks $a_0, a_1, a_2, ..., a_7$. The values $a'_0, a'_1, a'_2, ..., a'_7$ are in sparse form, and a' is a sparse a. We need the following lookup tables:

1. **SHA-256 NORMALIZE4**: Read a_i to a'_i
2. **SHA-512 8ROT1 64**: Read a'_0 to r_1
3. **SHA-512 8SHR7 64**: Read a'_0 to r_3

	w_1	w_2	w_3	w_4	w_o
j + 0	a_0	a_1	a_2	a_3	a_4
j + 1	a'_0	a'_1	a'_2	a'_3	a
j + 2	a_5	a_6	a_7	a'_4	σ_0
j + 3	a'_5	a'_6	a'_7	r_1	r_2

Sparse map gate constraints:

$$w_{o,j+1} = w_{1,j} + w_{2,j} \cdot 2^8 + w_{3,j} \cdot 2^{8\cdot 2} + w_{4,j} \cdot 2^{8\cdot 3} + w_{o,j} \cdot 2^{8\cdot 4} + w_{1,j+2} \cdot 2^{8\cdot 5} + w_{2,j+2} \cdot 2^{8\cdot 6} + w_{3,j+2} \cdot 2^{8\cdot 7}$$
$$w_{o,j+2} = w_{2,j+1} \cdot 4^{8-1} + w_{3,j+1} \cdot 4^{8\cdot 2-1} + w_{4,j+1} \cdot 4^{8\cdot 3-1} + w_{4,j+2} \cdot 4^{8\cdot 4-1} + w_{1,j+3} \cdot 4^{8\cdot 5-1} + w_{2,j+3} \cdot 4^{8\cdot 6-1} + w_{3,j+3} \cdot 4^{8\cdot 7-1} + w_{1,j+1} \cdot 4^{8\cdot 7} + w_{2,j+1} + w_{3,j+1} \cdot 4^8 + w_{4,j+1} \cdot 4^{8\cdot 2} + w_{4,j+2} \cdot 4^{8\cdot 3} + w_{1,j+3} \cdot 4^{8\cdot 4} + w_{2,j+3} \cdot 4^{8\cdot 5} + w_{3,j+3} \cdot 4^{8\cdot 6} + w_{2,j+1} \cdot 4^{8-7} + w_{3,j+1} \cdot 4^{8\cdot 2-7} + w_{4,j+1} \cdot 4^{8\cdot 3-7} + w_{4,j+2} \cdot 4^{8\cdot 4-7} + w_{1,j+3} \cdot 4^{8*5-7} + w_{2,j+3} \cdot 4^{8\cdot 6-7} + w_{3,j+3} \cdot 4^{8\cdot 7-7} + w_{4,j+3} + w_{o,j+3}$$
10 plookup constraints

The function σ_1 contain sparse mapping subcircuit with base 4. Let a be divided to 8 bits-chunks $a_0, a_1, a_2, ..., a_7$. The values $a'_0, a'_1, a'_2, ..., a'_7$ are in sparse form, and a' is a sparse a. We need the following lookup tables:

1. **SHA-256 NORMALIZE4**: Read a_i to a'_i
2. **SHA-512 8ROT3 64**: Read a'_2 to r_1
3. **SHA-512 8ROT5 SHR6 64**: Read $a'_7 + a'_0$ to r_2

	w_1	w_2	w_3	w_4	w_o
j + 0	a_0	a_1	a_2	a_3	a_4
j + 1	a'_0	a'_1	a'_2	a'_3	a
j + 2	a_5	a_6	a_7	a'_4	σ_1
j + 3	a'_5	a'_6	a'_7	r_1	r_2

Sparse map gate constraints:

$$w_{o,j+1} = w_{1,j} + w_{2,j} \cdot 2^8 + w_{3,j} \cdot 2^{8\cdot 2} + w_{4,j} \cdot 2^{8\cdot 3} + w_{o,j} \cdot 2^{8\cdot 4} + w_{1,j+2} \cdot 2^{8\cdot 5} + w_{2,j+2} \cdot 2^{8\cdot 6} + w_{3,j+2} \cdot 2^{8\cdot 7}$$
$$w_{o,j+2} = w_{1,j+1} \cdot 4^{64-19} + w_{2,j+1} \cdot 4^{64+(8-19)} + w_{4,j+1} \cdot 4^{8\cdot 3-19} + w_{4,j+2} \cdot 4^{8\cdot 4-19} + w_{1,j+3} \cdot 4^{8\cdot 5-19} + w_{2,j+3} \cdot 4^{8\cdot 6-19} + w_{3,j+3} \cdot 4^{8\cdot 7-19} + w_{1,j+1} \cdot 4^{64-61)} + w_{2,j+1} \cdot 4^{64+(8-61)} + w_{3,j+1} \cdot 4^{64+(8\cdot 2-61)} + w_{4,j+1} \cdot 4^{64+(8\cdot 3-61)} + w_{4,j+2} \cdot 4^{64+(8\cdot 4-61)} + w_{1,j+3} \cdot 4^{64+(8\cdot 5-61)} + w_{2,j+3} \cdot 4^{64+(8\cdot 6-61)} + w_{2,j+1} \cdot 4^{8-6} + w_{3,j+1} \cdot 4^{8\cdot 2-6} + w_{4,j+1} \cdot 4^{8\cdot 3-6} + w_{4,j+2} \cdot 4^{8\cdot 4-6} + w_{1,j+3} \cdot 4^{8\cdot 5-6} + w_{2,j+3} \cdot 4^{8\cdot 6-6} + w_{3,j+3} \cdot 4^{8\cdot 7-6} + w_{4,j+3} + w_{o,j+3}$$
10 plookup constraints

The sparse values σ_0 and σ_1 have to be normalized. The final addition requires one add gate. Note, that a' already initialized in the row $j-2$. We use **SHA256 NORMALIZE4**

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	acc
j + 1	a_0	a_1	a_2	a_3	0
j + 2	a'_4	a'_5	a'_6	a'_7	σ_i
j + 3	a_4	a_5	a_6	a_7	

Normalize gate constraints:

$$w_{o,j+2} = w_{4,j+1} \cdot 256^3 + w_{3,j+1} \cdot 256^2 + w_{2,j+1} \cdot 256 + w_{1,j+1} + w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7$$
$$w_{o,j} = w_{o,j-2} - (w_{4,j} \cdot 256^3 + w_{3,j} \cdot 256^2 + w_{2,j} \cdot 256 + w_{1,j})$$
$$w_{o,j+1} = w_{o,j} - (w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7)$$
8 plookup constraints

The Σ_0 **function** contain sparse mapping subcircuit with base 4. Let a be divided to 7-bits chunks a_0, a_1, a_2, a_3 and 9 bits-chunks a_4, a_5, a_6, a_7. The values $a'_0, a'_1, a'_2, ..., a'_7$ are in sparse form, and a' is a sparse a. We need the following lookup tables:

1. **SHA-512 9NORMALIZE4**: Read a_i to a'_i
2. **SHA-512 7NORMALIZE4**: Read a_i to a'_i
3. **SHA-512 9ROT6 64**: Read a'_4 to r_2
4. **SHA-512 9ROT2 64**: Read a'_5 to r_3

	w_1	w_2	w_3	w_4	w_o
j + 0	a_0	a_1	a_2	a_3	a_4
j + 1	a'_0	a'_1	a'_2	a'_3	a
j + 2	a_5	a_6	a_7	a'_4	Σ_0
j + 3	a'_5	a'_6	a'_7	r_1	r_2

Sparse map gate constraints:

$$w_{o,j+1} = w_{1,j} + w_{2,j} \cdot 2^7 + w_{3,j} \cdot 2^{7\cdot 2} + w_{4,j} \cdot 2^{7\cdot 3} + w_{o,j} \cdot 2^{7\cdot 4} + w_{1,j+2} \cdot 2^{7\cdot 4+9} + w_{2,j+2} \cdot 2^{7\cdot 4+9\cdot 2} + w_{3,j+2} \cdot 2^{7\cdot 4+9\cdot 3}$$
$$w_{o,j+2} = w_{4,j+2} + w_{1,j+3} \cdot 4^9 + w_{2,j+3} \cdot 4^{9\cdot 2} + w_{3,j+3} \cdot 4^{9\cdot 3} + w_{1,j+1} \cdot 4^{9\cdot 4} + w_{2,j+1} \cdot 4^{9\cdot 4+7}$$
$$+ w_{3,j+1} \cdot 4^{9\cdot 4+7\cdot 2} + w_{4,j+1} \cdot 4^{9\cdot 4+7\cdot 3} + w_{1,j+1} \cdot 4^{64-34)} + w_{2,j+1} \cdot 4^{64+(7-34)} + w_{3,j+1} \cdot 4^{64+(7\cdot 2-34)} +$$
$$w_{4,j+1} \cdot 4^{64+(7\cdot 3-34)} + w_{1,j+3} \cdot 4^{7\cdot 4+9-34} + w_{2,j+3} \cdot 4^{7\cdot 4+9\cdot 2-34} + w_{3,j+3} \cdot 4^{7\cdot 4+9\cdot 3-34} + w_{1,j+1} \cdot 4^{64-39)} +$$
$$w_{2,j+1} \cdot 4^{64+(7-39)} + w_{3,j+1} \cdot 4^{64+(7\cdot 2-39)} + w_{4,j+1} \cdot 4^{64+(7\cdot 3-39)} + w_{4,j+2} \cdot 4^{64+(7\cdot 4-39)} + w_{2,j+3} \cdot$$
$$4^{7\cdot 4+9\cdot 2-39} + w_{3,j+3} \cdot 4^{7\cdot 4+9\cdot 3-39} + w_{4,j+3} + w_{o,j+3}$$

10 plookup constraints

The Σ_1 function contain sparse mapping subcircuit with base 7. Let a be divided to 7-bits chunks a_0, a_1, a_2, a_3 and 9 bits-chunks a_4, a_5, a_6, a_7. The values $a'_0, a'_1, a'_2, ..., a'_7$ are in sparse form, and a' is a sparse a. We need the following lookup tables:

1. **SHA-512 9NORMALIZE7**: Read a_i to a'_i
2. **SHA-512 7NORMALIZE7**: Read a_i to a'_i
3. **SHA-512 7ROT4 32**: Read a'_2 to r_2
4. **SHA-512 9ROT4 32**: Read a'_5 to r_3

	w_1	w_2	w_3	w_4	w_o
j + 0	a_0	a_1	a_2	a_3	a_4
j + 1	a'_0	a'_1	a'_2	a'_3	a
j + 2	a_5	a_6	a_7	a'_4	Σ_1
j + 3	a'_5	a'_6	a'_7	r_1	r_2

Sparse map gate constraints:

$$w_{o,j+1} = w_{1,j} + w_{2,j} \cdot 2^7 + w_{3,j} \cdot 2^{7\cdot 2} + w_{4,j} \cdot 2^{7\cdot 3} + w_{o,j} \cdot 2^{7\cdot 4} + w_{1,j+2} \cdot 2^{7\cdot 4+9} + w_{2,j+2} \cdot 2^{7\cdot 4+9\cdot 2} + w_{3,j+2} \cdot 2^{7\cdot 4+9\cdot 3}$$
$$w_{o,j+2} = w_{3,j+1} + w_{4,j+1} \cdot 7^7 + w_{4,j+2} \cdot 7^{7\cdot 2} + w_{1,j+3} \cdot 7^{7\cdot 2+9} + w_{2,j+3} \cdot 7^{7\cdot 2+9\cdot 2} + w_{3,j+3} \cdot 7^{9\cdot 3+7\cdot 2} + w_{1,j+1} \cdot 7^{9\cdot 4+7\cdot 2} +$$
$$w_{2,j+1} \cdot 7^{9\cdot 4+7\cdot 3} + w_{1,j+1} \cdot 7^{64-18)} + w_{2,j+1} \cdot 7^{64+(7-18)} + w_{4,j+1} \cdot 7^{7\cdot 3-18} + w_{4,j+2} \cdot 7^{7\cdot 4-18} + w_{1,j+3} \cdot 7^{7\cdot 4+9-18} +$$
$$w_{2,j+3} \cdot 7^{7\cdot 4+9\cdot 2-18} + w_{3,j+3} \cdot 7^{7\cdot 4+9\cdot 3-18} + w_{1,j+1} \cdot 7^{64-41)} + w_{2,j+1} \cdot 7^{64+(7-41)} + w_{3,j+1} \cdot 7^{64+(7\cdot 2-41)} +$$
$$w_{4,j+1} \cdot 7^{64+(7\cdot 3-41)} + w_{4,j+2} \cdot 7^{64+(7\cdot 3+9-41)} + w_{2,j+3} \cdot 7^{64+(7\cdot 3+9\cdot 2-41)} + w_{3,j+3} \cdot 7^{7\cdot 3+9\cdot 3-41} + w_{4,j+3} + w_{o,j+3}$$

10 plookup constraints

The sparse values Σ_0 and Σ_1 have to be normalized. We use **SHA256 NORMALIZE4** and **SHA256 NORMALIZE7**. Note, that a' already initialized in the row $j-2$.

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	a''
j + 1	a_0	a_1	a_2	a_3	0
j + 2	a'_4	a'_5	a'_6	a'_7	Σ_i
j + 3	a_4	a_5	a_6	a_7	

Normalize gate constraints:

$$w_{o,j+2} = w_{4,j+1} \cdot 256^3 + w_{3,j+1} \cdot 256^2 + w_{2,j+1} \cdot 256 + w_{1,j+1} + w_{1,j+3} \cdot 256^4$$
$$+ w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7$$
$$w_{o,j} = w_{1,j-3} + w_{2,j-3} \cdot 4^7 + w_{3,j-3} \cdot 4^{7\cdot 2} + w_{4,j-3} \cdot 4^{7\cdot 3} + w_{4,j-2} \cdot 4^{7\cdot 4} + w_{1,j-1} \cdot 7^{7\cdot 4+9}$$
$$+ w_{2,j-1} \cdot 7^{7\cdot 4+9\cdot 2} + w_{2,j-1} \cdot 7^{7\cdot 4+9\cdot 3} \text{ for maj or ch function. For } \Sigma_1 \text{ replace 4 with 7}$$
$$w_{o,j+1} =$$
$$w_{o,j-2} - (w_{4,j} \cdot 256^3 + w_{3,j} \cdot 256^2 + w_{2,j} \cdot 256 + w_{1,j} + w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7)$$

8 plookup constraints

The Maj function contain sparse mapping subcircuit with base 4 for a, b, c. Note, that the sparse chunks of a we already have in Σ_0 in the circuit. The variables b and c were represented in sparse chunks in the previous rounds or it is public inputs.

	w_1	w_2	w_3	w_4	w_o
j	a'	b'	c'		maj

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + w_{2,j} + w_{3,j}$$

The sparse values maj have to be normalized. We use **SHA256 MAJ NORMALIZE4** Note, that the sparse maj already initialized in the row $j-1$.

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	acc
j + 1	a_0	a_1	a_2	a_3	0
j + 2	a'_4	a'_5	a'_6	a'_7	maj
j + 3	a_4	a_5	a_6	a_7	

Normalize gate constraints:

$$\begin{gathered} w_{o,j+2} = w_{4,j+1} \cdot 256^3 + w_{3,j+1} \cdot 256^2 + w_{2,j+1} \cdot 256 + w_{1,j+1} + w_{1,j+3} \cdot 256^4 \\ + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7 \\ w_{o,j} = w_{o,j-1} - (w_{4,j} \cdot 256^3 + w_{3,j} \cdot 256^2 + w_{2,j} \cdot 256 + w_{1,j}) \\ w_{o,j+1} = w_{o,j} - (w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7) \\ \text{8 plookup constraints} \end{gathered}$$

The final addition requires one add gate.

The Ch function contain sparse mapping subcircuit with base 7 for e, f, g. Note, that e we already have in the sparse from Σ_1 in the circuit. The variables f and g were represented in sparse form in the previous rounds or it is public inputs.

	w_1	w_2	w_3	w_4	w_o
j + 0	e'	f'	g'		ch

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + 2 \cdot w_{2,j} + 3 \cdot w_{3,j}$$

The sparse values ch have to be normalized. Note, that ch already initialized in the row $j-1$. We use **SHA256 CH NORMALIZE7**

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	acc
j + 1	a_0	a_1	a_2	a_3	0
j + 2	a'_4	a'_5	a'_6	a'_7	ch
j + 3	a_4	a_5	a_6	a_7	

Normalize gate constraints:

$$\begin{gathered} w_{o,j+2} = w_{4,j+1} \cdot 256^3 + w_{3,j+1} \cdot 256^2 + w_{2,j+1} \cdot 256 + w_{1,j+1} + w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 \\ + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7 \\ w_{o,j} = w_{o,j-1} - (w_{4,j} \cdot 256^3 + w_{3,j} \cdot 256^2 + w_{2,j} \cdot 256 + w_{1,j}) \\ w_{o,j+1} = w_{o,j} - (w_{1,j+3} \cdot 256^4 + w_{2,j+3} \cdot 256^5 + w_{3,j+3} \cdot 256^6 + w_{4,j+4} \cdot 256^7) \\ \text{8 plookup constraints} \end{gathered}$$

The final addition requires one add gate.
The updating of variables for new rounds costs 10 add gates.
Producing the final hash value costs two add gates.

2.5.4 Poseidon Circuit

Consider a poseidon permutation $F : [0_{\mathbb{F}}, I[2], I[3]] \rightarrow [O[1], H, O[3]]$ of width 3 and $\alpha = 5$. The 1-call sponge function is used:

	w_1	w_2	w_3	w_4	w_o
j + 0	$0_{\mathbb{F}}$	$I[2]$	$I[3]$	$T_{1,0}$	$T_{1,1}$
j + 1	$T_{1,2}$	$T_{2,0}$	$T_{2,1}$	$T_{2,2}$	$T_{3,0}$
...					
j + 39	$O[1]$	H	$O[3]$	–	–

Constraints:

For 4 rounds:
$$[w_{4,j}, w_{o,j}, w_{1,j+1}] = [w_{1,j}^5, w_{2,j}^5, w_{3,j}^5] \times M + RC$$
For 57 rounds:
$$[w_{1,j+4}, w_{2,j+4}, w_{3,j+4}] = [w_{3,j+3}, w_{4,j+3}, w_{o,j+3}^5] \times M + RC$$
For 4 rounds:
$$[w_{2,j+37}, w_{3,j+37}, w_{4,j+37}] = [w_{4,j+36}^5, w_{o,j+36}^5, w_{1,j+37}^5] \times M + RC$$

2.5.5 Merkle Tree Circuit

Merkle Tree generation for set $\{H_{B_{n_1}}, ..., H_{B_{n_2}}\}$. Let $k = \lceil \log(n_2 - n_1) \rceil$

1. $n = n_2 - n_1$
2. $2^k = n$
3. for i from 0 to $n - 1$:
   3.1 $T_i \coloneqq H_i$ // just notation for simplicity, not a real part of the circuit
4. for i from 0 to $k - 1$:
   4.1 for j from 0 to $(n - 1)/2$:
      4.1.1 $T'_i = \texttt{hash}(T_{2 \cdot i}, T_{2 \cdot i+1})$. // see Section 2.5.4
   4.2 $n = \frac{n}{2}$
   4.3 for j from 0 to $n - 1$:
      4.3.1 $T_i \coloneqq T'_i$. // just notation for simplicity, not a real part of the circuit

2.5.6 Ed25519 Circuit

To verify a signature (R, s) on a message M using public key A and a generator B do:

1. Prove that s in the range $L = 2^{252} + 27742317777372353535851937790883648493$.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	s	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
j + 1	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}	z_{16}
j + 2	z_{17}	z_{18}	z_{19}	z_{20}	z_{21}	z_{22}	z_{23}	z_{24}	z_{25}

   Constraints:

$$w_{2,j} = w_{1,j} + 2^{253} - L$$
Each $w_{i,k} - 2^{10} \cdot w_{i+a,k+b}$, where $i = 2, .., 9$ for $k = 0$, $i = 1, .., 9$ for $k = 1$ and $i = 1, .., 8$ for $k = 2$, $(i + 1) = b \cdot 9 + a$ is range-constrained by 10-bits plookup table.
$w_{9,j+2} \cdot 2^7$ is range-constrained by 10-bits plookup table.

   It costs 3 rows.

2. $k ==$ SHA-512$(data||R||A||M)$ // See section **??** It costs ? rows.
3. $sB ? = ? R + kA$:

3.1 Fixed-base scalar multiplication circuit is used for $sB = S$. The cell $w_{1,j+84}$ is copy-constrained with $w_{1,j+0}$ from the range circuit.

3.2 One addition is used for $S + (-R)$. The coordinates of R and $T = S + (-R)$ are placed on the last row of fixed-base scalar multiplication circuit. In total, three constraints are used for addition:

$$w_{4,j+84} \cdot (1 + dw_{7,j+84} \cdot (-w_{2,j+84}) \cdot w_{8,j+84} \cdot w_{3,j+84}) = w_{7,j+84} \cdot w_{3,j+84} + (-w_{2,j+84}) \cdot w_{8,j+84}$$
$$w_{5,j+84} \cdot (1 - dw_{7,j+84} \cdot (-w_{2,j+84}) \cdot w_{8,j+84} \cdot w_{3,j+84}) = w_{7,j+84} \cdot (-w_{2,j+84}) + w_{3,j+84} \cdot w_{8,j+84}$$
$$(-w_{2,j+84})^2 + w_{3,j+84}^2 = 1 - d \cdot w_{2,j+84}^2 \cdot w_{3,j+84}^2$$

3.3 Variable-base scalar multiplication circuit for $T = k \cdot A$, where cells $w_{2,j+254}, w_{3,j+254}$ are copy constrained with $w_{4,j+84}, w_{5,j+84}$ from the fixed-base scalar multiplication circuit.

It costs ...

2.5.7 Elliptic Curves Arithmetics

WIP

This section instantiates the arithmetic of edwards25519 curve:

$$-x^2 + y^2 = 1 - (121665/121666) \cdot x^2 \cdot y^2$$

Affine coordinates are used for points. Let d be equal to 121665/121666.

Fixed-base scalar multiplication circuit : We precompute all values $w(B, s, k) = k_i \cdot 8^s B$, where $k_i \in \{0, ..7\}$, $s \in \{0, .., 84\}$.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	acc	b_{252}	b_{251}	b_{250}	u_1	v_1	x_{acc}	y_{acc}	$--$
j + 1	acc	b_{249}	b_{248}	b_{247}	u_1	v_1	x_{acc}	y_{acc}	$--$
...									
j + 84	s	x_r	y_r	x_t	y_t	b_0	x_{acc}	y_{acc}	$--$

Define the following functions:

1. $\phi_1 : (x_1, x_2, x_3, x_4) \mapsto$
$x_3 \cdot (-u'_0 \cdot x_2 \cdot x_1 + u'_0 \cdot x_1 + u'_0 \cdot x_2 - u'_0 + u'_2 \cdot x_1 \cdot x_2 - u'_2 \cdot x_2 + u'_4 \cdot x_1 \cdot x_2 - u'_4 \cdot x_2 - u'_6 \cdot x_1 \cdot x_2 + u'_1 \cdot x_2 \cdot x_1 - u'_1 \cdot x_1 - u'_1 \cdot x_2 + u'_1 - u'_3 \cdot x_1 \cdot x_2 + u'_3 \cdot x_2 - u'_5 \cdot x_1 \cdot x_2 + u'_5 \cdot x_2 + u'_7 \cdot x_1 \cdot x_2) - (x_4 - u'_0 \cdot x_2 \cdot x_1 + u'_0 \cdot x_1 + u'_0 \cdot x_2 - u'_0 + u'_2 \cdot x_1 \cdot x_2 - u'_2 \cdot x_2 + u'_4 \cdot x_1 \cdot x_2 - u'_4 \cdot x_2 - u'_6 \cdot x_1 \cdot x_2)$

2. $\phi_2 : (x_1, x_2, x_3, x_4) \mapsto$
$x_3 \cdot (-v'_0 \cdot x_2 \cdot x_1 + v'_0 \cdot x_1 + v'_0 \cdot x_2 - v'_0 + v'_2 \cdot x_1 \cdot x_2 - v'_2 \cdot x_2 + v'_4 \cdot x_1 \cdot x_2 - v'_4 \cdot x_2 - v'_6 \cdot x_1 \cdot x_2 + v'_1 \cdot x_2 \cdot x_1 - v'_1 \cdot x_1 - v'_1 \cdot x_2 + v'_1 - v'_3 \cdot x_1 \cdot x_2 + v'_3 \cdot x_2 - v'_5 \cdot x_1 \cdot x_2 + v'_5 \cdot x_2 + v'_7 \cdot x_1 \cdot x_2) - (x_4 - v'_0 \cdot x_2 \cdot x_1 + v'_0 \cdot x_1 + v'_0 \cdot x_2 - v'_0 + v'_2 \cdot x_1 \cdot x_2 - v'_2 \cdot x_2 + v'_4 \cdot x_1 \cdot x_2 - v'_4 \cdot x_2 - v'_6 \cdot x_1 \cdot x_2)$

3. $\phi_3 : (x_1, x_3, x_4, x_5, x_6) \mapsto$
$x_1 \cdot (1 + d \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6) - (x_3 \cdot x_6 + x_4 \cdot x_5)$

4. $\phi_4 : (x_2, x_3, x_4, x_5, x_6) \mapsto$
$x_2 \cdot (1 - d \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6) - (x_3 \cdot x_5 + x_4 \cdot x_6)$

Constraints:

- For $j + 0$:
  - $(w_{2,j+0} - 1) \cdot w_{2,j+0} = 0$
  - $(w_{3,j+0} - 1) \cdot w_{3,j+0} = 0$
  - $(w_{4,j+0} - 1) \cdot w_{4,j+0} = 0$
  - $w_{1,j+0} = w_{2,j+0} \cdot 2^2 + w_{3,j+0} \cdot 2 + w_{4,j+0}$
  - $\phi_1(w_{2,j+0}, w_{3,j+0}, w_{4,j+0}, w_{5,j+0}) = 0$, where $(u'_i, v'_i) = w(B, j + 0, i)$
  - $\phi_2(w_{2,j+0}, w_{3,j+0}, w_{4,j+0}, w_{6,j+0}) = 0$, where $(u'_i, v'_i) = w(B, j + 0, i)$
  - $w_{7,j+0} = w_{5,j+0}$
  - $w_{8,j+0} = w_{6,j+0}$
- For $j + z$, $z \neq 0$, $z \neq 84$:

- $(w_{2,j+z} - 1) \cdot w_{2,j+z} = 0$
- $(w_{3,j+z} - 1) \cdot w_{3,j+z} = 0$
- $(w_{4,j+z} - 1) \cdot w_{4,j+z} = 0$
- $w_{1,j+z} = w_{1,j+z-1} \cdot 2^3 + w_{2,j+z} \cdot 2^2 + w_{3,j+z} \cdot 2 + w_{4,j+z}$
- $\phi_1(w_{2,j+z}, w_{3,j+z}, w_{4,j+z}, w_{5,j+z}) = 0$, where $(u'_i, v'_i) = w(B, j+z, i)$
- $\phi_2(w_{2,j+z}, w_{3,j+z}, w_{4,j+z}, w_{6,j+z}) = 0$, where $(u'_i, v'_i) = w(B, j+z, i)$
- $\phi_3(w_{7,j+z}, w_{7,j+z-1}, w_{8,j+z-1}, w_{5,j+z}, w_{6,j+z}) = 0$
- $\phi_4(w_{8,j+z}, w_{7,j+z-1}, w_{8,j+z-1}, w_{5,j+z}, w_{6,j+z}) = 0$

- For $j + 84$:
  - $(w_{6,j+84} - 1) \cdot w_{6,j+84} = 0$
  - $w_{1,j+84} = w_{1,j+83} \cdot 2 + w_{6,j+84}$
  - $\phi_3(w_{7,j+z}, w_{7,j+z-1}, w_{8,j+z-1}, w_{6,j+84} \cdot x_B, w_{6,j+84} \cdot y_B + (1 - w_{6,j+84})) = 0$
  - $\phi_4(w_{8,j+z}, w_{7,j+z-1}, w_{8,j+z-1}, w_{6,j+84} \cdot x_B, w_{6,j+84} \cdot y_B + (1 - w_{6,j+84})) = 0$, where $B = (x_B, y_B)$.

Variable-base scalar multiplication circuit :

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
j + 0	acc	b_{511}	b_{510}	x_2	y_2	b_{509}	x_3	y_3	b_{508}
j + 1	acc	x_1	y_1	x_4	y_4	b_{507}	x_5	y_5	b_{506}
j + 2	acc	$--$	$--$	x_6	y_6	b_{505}	x_7	y_7	b_{504}
...									
j + 253	acc	x_1	y_1	x_4	y_4	b_3	x_3	y_3	b_2
j + 254	k	x_0	y_0	x_2	y_2	b_1	x_1	y_1	b_0

Define the following functions:

1. $\phi_1 : (b, x_1, y_1, x_2, y_2, x_3) \mapsto$
   $x_3 \cdot ((y_1^2 - x_1^2) \cdot (2 - y_1^2 + x_1^2) + 2dx_1y_1(y_1^2 + x_1^2) \cdot x_2y_2b) - (2x_1y_1 \cdot (2 - y_1^2 + x_1^2) \cdot (y_2b + (1 - b)) + (y_1^2 + x_1^2) \cdot (y_1^2 - x_1^2) \cdot x_2b)$

2. $\phi_2 : (b, x_1, y_1, x_2, y_2, y_3) \mapsto$
   $y_3 \cdot ((y_1^2 - x_1^2) \cdot (2 - y_1^2 + x_1^2) - 2dx_1y_1(y_1^2 + x_1^2) \cdot x_2y_2b) - (2x_1y_1 \cdot (2 - y_1^2 + x_1^2) \cdot x_2b + (y_1^2 + x_1^2) \cdot (y_1^2 - x_1^2) \cdot (y_2b + (1 - b)))$

Constraints:

- For $j + 0$:
  - $(w_{2,j+0} - 1) \cdot w_{2,j+0} = 0$
  - $(w_{3,j+0} - 1) \cdot w_{3,j+0} = 0$
  - $(w_{6,j+0} - 1) \cdot w_{6,j+0} = 0$
  - $(w_{9,j+0} - 1) \cdot w_{9,j+0} = 0$
  - $w_{1,j+0} = w_{2,j} \cdot 2^3 + w_{3,j+0} \cdot 2^2 + w_{6,j+0} \cdot 2 + w_{9,j+0}$
  - $\phi_1(w_{3,j+0}, w_{2,j+1} \cdot w_{2,j+0}, (w_{3,j+1} \cdot w_{2,j+0} + (1 - w_{2,j+0})), w_{2,j+1}, w_{3,j+1}, w_{4,j+0})$
  - $\phi_2(w_{3,j+0}, w_{2,j+1} \cdot w_{2,j+0}, (w_{3,j+1} \cdot w_{2,j+0} + (1 - w_{2,j+0})), w_{2,j+1}, w_{3,j+1}, w_{5,j+0})$
  - $\phi_1(w_{6,j+0}, w_{4,j+0}, (w_{5,j+0}, w_{2,j+1}, w_{3,j+1}, w_{7,j+0})$
  - $\phi_2(w_{6,j+0}, w_{4,j+0}, (w_{5,j+0}, w_{2,j+1}, w_{3,j+1}, w_{8,j+0})$
- For $j + z$, $z \equiv 1 \mod 2$:
  - $(w_{6,j+z} - 1) \cdot w_{6,j+z} = 0$
  - $(w_{9,j+z} - 1) \cdot w_{9,j+z} = 0$
  - $w_{1,j+z} = w_{1,j+z-1} \cdot 2^2 + w_{6,j+z} \cdot 2 + w_{9,j+z}$
  - $\phi_1(w_{9,j+z-1}, w_{7,j+z-1}, w_{8,j+z-1}, w_{2,j+z}, w_{3,j+z}, w_{4,j+z})$
  - $\phi_2(w_{9,j+z-1}, w_{7,j+z-1}, w_{8,j+z-1}, w_{2,j+z}, w_{3,j+z}, w_{5,j+z})$
  - $\phi_1(w_{6,j+z}, w_{4,j+z}, w_{5,j+z}, w_{2,j+z}, w_{3,j+z}, w_{7,j+z})$
  - $\phi_2(w_{6,j+z}, w_{4,j+z}, w_{5,j+z}, w_{2,j+z}, w_{3,j+z}, w_{8,j+z})$
- For $j + z$, $z \equiv 0 \mod 2, z \neq 0$:
  - $(w_{6,j+z} - 1) \cdot w_{6,j+z} = 0$
  - $(w_{9,j+z} - 1) \cdot w_{9,j+z} = 0$
  - $w_{1,j+z} = w_{1,j+z-1} \cdot 2^2 + w_{6,j+z} \cdot 2 + w_{9,j+z}$

- $\phi_1(w_{9,j+z-1}, w_{7,j+z-1}, w_{8,j+z-1}, w_{2,j+z-1}, w_{3,j+z-1}, w_{4,j+z})$
- $\phi_2(w_{9,j+z-1}, w_{7,j+z-1}, w_{8,j+z-1}, w_{2,j+z-1}, w_{3,j+z-1}, w_{5,j+z})$
- $\phi_1(w_{6,j+z}, w_{4,j+z}, w_{5,j+z}, w_{2,j+z-1}, w_{3,j+z-1}, w_{7,j+z})$
- $\phi_2(w_{6,j+z}, w_{4,j+z}, w_{5,j+z}, w_{2,j+z-1}, w_{3,j+z-1}, w_{8,j+z})$

- For $j + 254$:
  - $(w_{6,j+z} - 1) \cdot w_{6,j+z} = 0$
  - $(w_{9,j+z} - 1) \cdot w_{9,j+z} = 0$
  - $w_{1,j+254} = w_{1,j+253} \cdot 2^2 + w_{6,j+254} \cdot 2 + w_{9,j+254}$
  - $\phi_1(w_{9,j+253}, w_{7,j+253}, w_{8,j+253}, w_{2,j+253}, w_{3,j+253}, w_{2,j+254})$
  - $\phi_2(w_{9,j+253}, w_{7,j+253}, w_{8,j+253}, w_{2,j+253}, w_{3,j+253}, w_{5,j+254})$
  - $\phi_1(w_{6,j+254}, w_{4,j+254}, w_{5,j+254}, w_{2,j+253}, w_{3,j+253}, w_{7,j+254})$
  - $\phi_2(w_{6,j+254}, w_{4,j+254}, w_{5,j+254}, w_{2,j+253}, w_{3,j+253}, w_{8,j+254})$
  - $\phi_1(w_{9,j+254}, w_{7,j+254}, w_{8,j+254}, w_{2,j+253}, w_{3,j+253}, w_{2,j+254})$
  - $\phi_2(w_{9,j+254}, w_{7,j+254}, w_{8,j+254}, w_{2,j+253}, w_{3,j+253}, w_{3,j+254})$

2.5.8 Redshift Verification

WIP

Redshift circuit repeats all steps from Section 2.4.2. The verification circuit is a part of bridge design, and it is supposed that any output of the basic proof is an input to the verification circuit. Thus, we do not suppose any decoding for the proof because it can be represented directly in the desirable form.

In the previous sections, we described circuits for most of the steps of the verifier algorithm. However, steps 15-16 require additional clarification.

We consider step 16 firstly as a simpler one. It contains basic arithmetic operations over finite field elements. These operations can be done with standard generic PLONK gate:

$$\mathbf{q}_L \cdot w_0 + \mathbf{q}_R \cdot w_1 + \mathbf{q}_M \cdot w_0 \cdot w_1 + \mathbf{q}_O \cdot w_2 + \mathbf{q}_C$$

There are more optimal ways to perform these calculations. However, the number of arithmetic operations is much less than in Step 15. It means that any optimizations do not decrease prover or verifier complexities in any noticeable way.

FRI Verification is the main part of Step 15. It contains two operations: Merkle tree path check and polynomial interpolation. The circuit version of Merkle path check algorithm does not differ from the original one. The circuit form Section 2.5.4 is used to check hash operations correctness.

To check polynomial interpolation, the following circuit is used:

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
j + 0	a_0	a_1	s_0	s_1	x	y	α	β	...

Constraints (**max degree** = 2):

1. $w_6 \cdot w_0 + w_7 = w_2 \longleftrightarrow \alpha \cdot a_0 + \beta = s_0$
2. $w_6 \cdot w_0 + w_7 = w_2 \longleftrightarrow \alpha \cdot a_1 + \beta = s_1$
3. $w_6 \cdot w_0 + w_7 = w_2 \longleftrightarrow \alpha \cdot x + \beta = y$

Copy constraints:

1. a_0, a_1, s_0, s_1, y are constrained by public input.

The gate uses the line equation to check that all three points are on the same line. This means, it checks $f(a_0) = s_0$, $f(a_1) = s_1$, $f(x) = y$ for $f(X) = \alpha \cdot X + \beta$.

2.5.9 Validator Set Proof Circuit

WIP

Chapter 3

In-EVM State Proof Verifier

This introduces a description for Solana's 'Light-Client' state proof in-EVM verifier. Crucial components which define this part design are:

1. Verification architecture description.
2. Verification logic API reference.
3. Input data structures description.

3.1 Verification Logic Architecture

3.2 Verification Logic API Reference

3.3 Input Data Structures

Bibliography

1. Kattis A., Panarin K., Vlasov A. RedShift: Transparent SNARKs from List Polynomial Commitment IOPs. Cryptology ePrint Archive, Report 2019/1400. 2019. https://ia.cr/2019/1400.

2. Gabizon A., Williamson Z. J., Ciobotaru O. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. 2019. https://ia.cr/2019/953.

3. Fast Reed-Solomon interactive oracle proofs of proximity / E. Ben-Sasson, I. Bentov, Y. Horesh et al. // 45th international colloquium on automata, languages, and programming (icalp 2018) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

4. Gabizon A., Williamson Z. J. Proposal: The Turbo-PLONK program syntax for specifying SNARK programs. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.

5. PLONKish Arithmetization - The halo2 book. https://zcash.github.io/halo2/concepts/arithmetization.html.

6. Gabizon A., Williamson Z. J. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315. 2020. https://ia.cr/2020/315.

7. Lookup argument - The halo2 book. https://zcash.github.io/halo2/design/proving-system/lookup.html.

8. Chiesa A., Ojha D., Spooner N. Fractal: Post-Quantum and Transparent Recursive Proofs from Holography. Cryptology ePrint Archive, Report 2019/1076. 2019. https://ia.cr/2019/1076.