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)

November 17, 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. Define polynomials ($F_0, \ldots, F_4$ - copy-satisfability):

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

17. For the lookup:

17.1 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.

17.2 $F_6(X) = L_0(X)(1 - V(X))$

17.3 $F_7(X) = q_{last} \cdot (V(X)^2 - V(X))$

17.4 $F_8(X) = (1 - (q_{last} + q_{blind})) \cdot (V(\omega X)(A'(X) + \beta)(S'(X) + \gamma) - V(X)(\theta^{m-1}A_0(X) + ... + A_{m-1}(X) + \beta)(\theta^{m-1}S_0(X) + ... + S_{m-1}(X) + \gamma))$

17.5 $F_9(X) = L_0(X) \cdot (A'(X) - S'(X))$

17.6 $F_{10}(X) = (1 - (q_{last} + q_{blind})) \cdot (A'(X) - S'(X)) \cdot (A'(X) - A'(\omega^{-1}X))$

18. Compute:

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

19. Split $T(X)$ into separate polynomials $T_0(X), ..., T_{N_{\texttt{perm}}-1}(X)$
20. Add commitments to $T_0(X), ..., T_{N_{\texttt{perm}}-1}(X)$ to transcript
21. Get $y \in \mathbb{F}/H$ from $hash|_{\mathbb{F}/H}$(transcript)
22. Run evaluation scheme with the committed polynomials and y
    Remark: Depending on the circuit, evaluation can be done also on $y\omega, y\omega^{-1}$.
23. The proof is $\pi_{\texttt{comm}}$ and $\pi_{\texttt{eval}}$„ where:
    - $\pi_{\texttt{comm}} = \{f_{0,\texttt{comm}}, \ldots, f_{N_{\texttt{wires}}-1,\texttt{comm}}, P_{\texttt{comm}}, Q_{\texttt{comm}}, T_{0,\texttt{comm}}, ..., T_{N_{\texttt{perm}}-1,\texttt{comm}}, A'_{\texttt{comm}}, S'_{\texttt{comm}}, V_{\texttt{comm}}\}$
    - $\pi_{\texttt{eval}}$ is evaluation proofs for $f_0(y), \ldots, f_{N_{\texttt{wires}}}(y), P(y), P(y\omega), Q(y), Q(y\omega)$, $T_0(y), \ldots, T_{N_{\texttt{perm}}-1}(y), A'(y), A'(y\omega^{-1}), S'(y), V(y), V(y\omega)$

2.4.2 Verifier View

1. Let $f_{0,\texttt{comm}}, \ldots, f_{N_{\texttt{wires}}-1,\texttt{comm}}$ be commitments to $f_0(X), \ldots, f_{N_{\texttt{wires}}-1}(X)$
2. transcript = setup_values$||f_{0,\texttt{comm}}|| \ldots ||f_{N_{\texttt{wires}}-1,\texttt{comm}}$
3. $\theta = hash$(transcript)
4. Let $A'_{\texttt{comm}}, S'_{\texttt{comm}}$ be commitments to $A'(X), S'(X)$.
5. transcript = transcript$||A'_{\texttt{comm}}||S'_{\texttt{comm}}$
6. $\beta, \gamma = hash$(transcript)
7. Let $P_{\texttt{comm}}, Q_{\texttt{comm}}, V_{i,\texttt{comm}}$ be commitments to $P(X), Q(X), V(X)$.
8. transcript = transcript$||P_{\texttt{comm}}||Q_{\texttt{comm}}||V_{\texttt{comm}}$
9. $\alpha_0, \ldots, \alpha_5 = hash$(transcript)
10. Let $T_{0,\texttt{comm}}, ..., T_{N_{\texttt{perm}}-1,\texttt{comm}}$ be commitments to $T_0(X), ..., T_{N_{\texttt{perm}}-1}(X)$
11. transcript = transcript$||T_{0,\texttt{comm}}||...||T_{N_{\texttt{perm}}-1,\texttt{comm}}$
12. $y = hash_{\mathbb{F}/H}$(transcript)
13. 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.
14. Calculate:

$$F_0(y) = L_1(y)(P(y) - 1)$$
$$F_1(y) = L_1(y)(Q(y) - 1)$$
$$p'(y) = \prod p_i(y) = \prod f_i(y) + \beta \cdot S_{id_i}(y) + \gamma$$
$$F_2(y) = P(y)p'(y) - P(y\omega)$$
$$q'(y) = \prod q_i(y) = \prod f_i(y) + \beta \cdot S_{\sigma_i}(y) + \gamma$$
$$F_3(y) = Q(y)q'(y) - Q(y\omega)$$
$$F_4(y) = L_n(y)(P(y\omega) - Q(y\omega))$$
$$F_5(y) = \sum_{0 \le i < N_{\texttt{sel}}} (\mathbf{q}_i(y) \cdot \texttt{gate}_i(y)) + \sum_{0 \le i < N_{\texttt{const}}} (\mathbf{f}_{c_i}(y)) + PI(y)$$
$$T(y) = \sum_{0 \le j < N_{\texttt{perm}+1}} y^{n \cdot j} T_j(y) \quad F_6(y) = L_0(y)(1 - V(y))$$
$$F_7(y) = q_{last} \cdot (V(y)^2 - V(y))$$
$$F_8(y) = (1 - (q_{last} + q_{blind})) \cdot (V(\omega y)(A'(y) + \beta)(S'(y) + \gamma) - V(y)(\theta^{m-1}A_0(y) + ... + A_{m-1}(y) + \beta)(\theta^{m-1}S_{i,0}(y) + ... + S_{m-1}(y) + \gamma))$$
$$F_9(y) = L_0(y) \cdot (A'(y) - S'(y))$$
$$F_{10}(y) = (1 - (q_{last} + q_{blind})) \cdot (A'(y) - S'(y)) \cdot (A'(y) - A'(\omega^{-1}y))$$

15. 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[2].

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 2.5.8
3. for i from $n_1 + 1$ to $n_2 + 32$:

   3.1 $H_{B_i} =$ `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 SHA2-256 Circuit

Suppose that input data in the 32-bits form, which is already padded to the required size. Checking that chunked input data corresponds to the original data out of this circuit. However, we add the boolean check and range proof.

Range proof that $a < 2^{32}$ Let $a = \{a_0, ..., a_{15}\}$, where a_i is two bits.

	w_1	w_2	w_3	w_4	w_o
j + 0	a_{12}	a_{13}	a_{14}	a_{15}	acc
j + 1	a_8	a_9	a_{10}	a_{11}	acc
j + 2	a_4	a_5	a_6	a_7	acc
j + 3	a_0	a_1	a_2	a_3	a

Range gate constraints:

$$w_{1,i}(w_{1,i}-1)(w_{1,i}-2)(w_{1,i}-3) + w_{2,i}(w_{2,i}-1)(w_{2,i}-2)(w_{2,i}-3) + w_{3,i}(w_{3,i}-1)(w_{3,i}-2)(w_{3,i}-3) + w_{4,i}(w_{4,i}-1)(w_{4,i}-2)(w_{4,i}-3) = 0$$
$$w_{o,i} = w_{o,i-1} \cdot 4^4 + w_{4,i} \cdot 4^3 + w_{3,i} \cdot 4^2 + w_{2,i} \cdot 4 + w_{1,i}$$

The range proofs are included for each input data block.

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

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_3. The values a'_0, a'_1, a'_2, a'_3 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-256 8ROT3 32**: Read a'_1 to r_1
3. **SHA-256 8ROT2 32**: Read a'_4 to r_2
4. **SHA-256 8SHR3 32**: 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
j + 1	a'_0	a'_1	a'_2	a'_3	acc
j + 2	$r1$	r_2	r_3		σ_0

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + w_{2,j} \cdot 2^8 + w_{3,j} \cdot 2^{8\cdot 2} + w_{4,j} \cdot 2^{8\cdot 3}$$
$$w_{o,j+1} = w_{2,j+1} \cdot 4^{8-7} + w_{3,j+1} \cdot 4^{8\cdot 2-7} + w_{4,j+1} \cdot 4^{8\cdot 3-7} + w_{1,j+1} \cdot 4^{8\cdot 2-2} + w_{2,j+1} \cdot 4^{8\cdot 3-2}$$
$$+w_{4,j+1} \cdot 4^{8-2} + w_{2,j+1} \cdot 4^{8-3} + w_{3,j+1} \cdot 4^{8\cdot 2-3} + w_{4,j+1} \cdot 4^{8^3-3}$$
$$w_{o,j+2} = w_{0,j+1} + w_{1,j+2} + w_{2,j+2} + w_{3,j+2}$$
7 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_3. The values a'_0, a'_1, a'_2, a'_3 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-256 8ROT1 32**: Read a'_2 to r_1
3. **SHA-256 8ROT3 32**: Read a'_2 to r_2
4. **SHA-256 8ROT2 32**: Read a'_1 to r_3

	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	acc
j + 2	$r1$	r_2	r_3		σ_1

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + w_{2,j} \cdot 2^8 + w_{3,j} \cdot 2^{8\cdot 2} + w_{4,j} \cdot 2^{8\cdot 3}$$
$$w_{o,j+1} = w_{1,j+1} \cdot 4^{8\cdot 2-1} + w_{2,j+1} \cdot 4^{8\cdot 3-1} + w_{4,j+1} \cdot 4^{8-1} + w_{1,j+1} \cdot 4^{8\cdot 2-3} + w_{2,j+1} \cdot 4^{8\cdot 3-3}$$
$$+w_{4,j+1} \cdot 4^{8-3} + w_{1,j+1} \cdot 4^{8\cdot 3-2} + w_{3,j+1} \cdot 4^{8-2} + w_{4,j+1} \cdot 4^{8^2-2}$$
$$w_{o,j+2} = w_{0,j+1} + w_{1,j+2} + w_{2,j+2} + w_{3,j+2}$$
7 plookup constraints

The sparse values σ_0 and σ_1 have to be normalized. The final addition requires one add gate. We use **SHA256 NORMALIZE4**

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	
j + 1	a_0	a_1	a_2	a_3	σ_i

Normalize gate constraints:

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

The Σ_0 function contain sparse mapping subcircuit with base 2. Let a be divided to 8 bits-chunks a_0, a_1, a_2, a_3. The values a'_0, a'_1, a'_2, a'_3 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-256 8ROT2 32**: Read a'_0 to r_1
3. **SHA-256 8ROT5 32**: Read a'_1 to r_2
4. **SHA-256 8ROT6 32**: Read a'_2 to r_3

	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	a'
j + 2	$r1$	r_2	r_3		Σ_0

Sparse map gate constraints:

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

The Σ_1 function contain sparse mapping subcircuit with base 2. Let a be divided to 8 bits-chunks a_0, a_1, a_2, a_3. The values a'_0, a'_1, a'_2, a'_3 are in sparse form, and a' is a sparse a. We need the following lookup tables:

1. **SHA-256 NORMALIZE7**: Read a_i to a'_i
2. **SHA-256 8ROT6 32**: Read a'_0 to r_1
3. **SHA-256 8ROT3 32**: Read a'_1 to r_2
4. **SHA-256 8ROT1 32**: Read a'_3 to r_3

	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	a'
j + 2	$r1$	r_2	r_3		Σ_1

Sparse map gate constraints:

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

The sparse values Σ_0 and Σ_1 have to be normalized. We use **SHA256 NORMALIZE4** and **SHA256 NORMALIZE7**.

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	
j + 1	a_0	a_1	a_2	a_3	Σ_i

Normalize gate constraints:

$$w_{o,j-1} = w_{4,j} \cdot 4^8 \cdot 3 + w_{3,j} \cdot 4^8 \cdot 2 + w_{2,j} \cdot 4^8 + w_{1,j} \text{ for } \Sigma_1 \text{ replace 4 with 7}$$
$$w_{o,i} = w_{4,i} \cdot 256^3 + w_{3,i} \cdot 256^2 + w_{2,i} \cdot 256 + w_{1,i}$$
7 plookup constraints

The Maj function contain sparse mapping subcircuit with base 2 for a, b, c. Let $a; b; c$ be divided to 8 bits-chunks $a_0, a_1, a_2, a_3; b_0, b_1, b_2, b_3; c_0, c_1, c_2, c_3$. The values a'_0, a'_1, a'_2, a'_3 are in sparse form, and a' is a sparse a. Similarly for b and c. Note, that a we already have in the sparse from Σ_0 in the circuit. The variables b and c 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 - k	a'_0	a'_1	a'_2	a'_3	a'
...					
j - l	b'_0	b'_1	b'_2	b'_3	b'
...					
j - t	c'_0	c'_1	c'_2	c'_3	c'
...					
j + 0	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**

	w_1	w_2	w_3	w_4	w_o
j + 0	a'_0	a'_1	a'_2	a'_3	
j + 1	a_0	a_1	a_2	a_3	maj

Normalize gate constraints:

$$w_{o,i} = w_{4,i} \cdot 256^3 + w_{3,i} \cdot 256^2 + w_{2,i} \cdot 256 + w_{1,i}$$

The final addition requires one add gate.

The Ch function contain sparse mapping subcircuit with base 2 for e, f, g. Let $e; f; g$ be divided to 8 bits-chunks $e_0, e_1, e_2, e_3; f_0, f_1, f_2, f_3; g_0, g_1, g_2, g_3$. The values e'_0, e'_1, e'_2, e'_3 are in sparse form, and e' is a sparse e. Similarly for b and c. 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 - k	a'_0	a'_1	a'_2	a'_3	a'
...					
j - l	b'_0	b'_1	b'_2	b'_3	b'
...					
j - t	c'_0	c'_1	c'_2	c'_3	c'
...					
j + 0	a'	b'	c'		ch

Sparse map gate constraints:

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

The sparse values ch have to be normalized. 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	
j + 1	a_0	a_1	a_2	a_3	ch

Normalize gate constraints:

$$w_{o,i} = w_{4,i} \cdot 256^3 + w_{3,i} \cdot 256^2 + w_{2,i} \cdot 256 + w_{1,i}$$

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.

2.5.3 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:

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

8 plookup constraints

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:

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

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 $F : [0_{\mathbb{F}}, I[2], I[3]] \to [O[1], H, O[3]]$ 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:
$$[w_{4,j}, w_{o,j}, w_{1,j+1}] = [w^5_{1,j}, w^5_{2,j}, w^5_{3,j}] \times M + RC$$
For 57 rounds:
$$[w_{1,j+4}, w_{2,j+4}, w_{3,j+4}] = [w_{3,j+3}, w_{4,j+3}, w^5_{o,j+3}] \times M + RC$$
For 4 rounds:
$$[w_{2,j+37}, w_{3,j+37}, w_{4,j+37}] = [w^5_{4,j+36}, w^5_{o,j+36}, w^5_{1,j+37}] \times M + RC$$

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_o
j + 0	s	z_0	z_1	z_2	z_3
...					
j + 5	z_{25}			$--$	

   Constraints:

$$w_{2,j} = w_{1,j} + 2^{253} - L$$

   Each $w_{i,k} - 2^{10} \cdot w_{next}$, where $i = 2, .., o$ for $k = 0$ and $i = 1, .., o$ for $k = 1, .., 4$ is range-constrained by 10-bits plookup table.

$$w_{1,j+5} \cdot 2^7 \text{ is range-constrained by 10-bits plookup table.}$$

2. $k ==$ SHA-512$(data||R||A||M)$ // See section **??**
3. $sB ? = ? R + kA$:
   3.1 Fixed-base scalar multiplication circuit is used for $sB = S$
   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:

$$x_t \cdot (1 + dx_s \cdot (-x_r) \cdot y_s \cdot y_r) = x_s \cdot y_r + (-x_r) \cdot y_s$$
$$y_t \cdot (1 - dx_s \cdot (-x_r) \cdot y_s \cdot y_r) = x_s \cdot (-x_r) + y_r \cdot y_s$$
$$-x_r^2 + y_r^2 = 1 - d \cdot x_r^2 \cdot y_r^2$$

   3.3 Variable-base scalar multiplication circuit has to be used in reversed order, where $(x_n, y_n) = (x_t, y_t)$.

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_o
j + 0	b_{n-1}	b_{n-2}	b_{n-3}	u_1	acc
j + 1	x_2	y_2	b_{n-6}	u_2	v_1
j + 2	b_{n-4}	b_{n-5}	v_2	b_{n-7}	acc
j + 3	x_3	y_3	b_{n-8}	b_{n-9}	u_3
j + 4	x_4	y_4	v_3	$-$	acc
...					
j + 84	$-$	$-$	v_{85}	$-$	$-$

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. $\phi_3 : (x_1, x_2, x_3, x_4, x_5, x_6) \mapsto$
$x_1 \cdot (1 + d \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6) - (x_3 \cdot x_6 + x_4 \cdot x_5)$

4. $\phi_4 : (x_1, x_2, x_3, x_4, x_5, x_6) \mapsto$
$x_2 \cdot (1 - d \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6) - (x_3 \cdot x_5 + x_4 \cdot x_6)$

Constraints:

- For $j + 0$:
  - $w_{o,j} = w_{1,j} \cdot 2^2 + w_{2,j} \cdot 2 + w_{3,j}$
  - $\phi_3(w_{1,j+1}, w_{2,j+1}, w_{4,j}, w_{o,j+1}, w_{4,j+1}, w_{3,j+2}) = 0$
  - $\phi_4(w_{1,j+1}, w_{2,j+1}, w_{4,j}, w_{o,j+1}, w_{4,j+1}, w_{3,j+2}) = 0$
- For $j + z$, $z \equiv 0 \mod 5$, $z \neq 0$:
  - $w_{o,j+z} = w_{1,j+z} \cdot 2^2 + w_{2,j+z} \cdot 2 + w_{3,j+z} + w_{o,j+z-1} \cdot 2^3$
  - $\phi_1(w_{1,j+z}, w_{2,j+z}, w_{3,j+z}, w_{4,j+z}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z}{5}), i)$
  - $\phi_2(w_{1,j+z}, w_{2,j+z}, w_{3,j+z}, w_{o,j+z+1}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z}{5}), i)$
  - $\phi_3(w_{1,j+z+1}, w_{2,j+z+1}, w_{1,j+z-1}, w_{2,j+z-1}, w_{4,j+z+1}, w_{3,j+z+2}) = 0$
  - $\phi_4(w_{1,j+z+1}, w_{2,j+z+1}, w_{1,j+z-1}, w_{2,j+z-1}, w_{4,j+z+1}, w_{3,j+z+2}) = 0$
- For $j + z$, $z \equiv 2 \mod 5$:
  - $w_{o,j+z} = w_{1,j+z} \cdot 2^2 + w_{2,j+z} \cdot 2 + w_{3,j+z-1} + w_{o,j+z-2} \cdot 2^3$
  - $\phi_1(w_{1,j+z}, w_{2,j+z}, w_{3,j+z-1}, w_{4,j+z-1}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z-2}{5}) + 1, i)$
  - $\phi_2(w_{1,j+z}, w_{2,j+z}, w_{3,j+z-1}, w_{3,j+z}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z-2}{5}) + 1, i)$
  - $\phi_3(w_{1,j+z+1}, w_{2,j+z+1}, w_{1,j+z-1}, w_{2,j+z-1}, w_{o,j+z+1}, w_{3,j+z+2}) = 0$
  - $\phi_4(w_{1,j+z+1}, w_{2,j+z+1}, w_{1,j+z-1}, w_{2,j+z-1}, w_{o,j+z+1}, w_{3,j+z+2}) = 0$
- For $j + z$, $z \equiv 3 \mod 5$:
  - $\phi_1(w_{4,j+z-1}, w_{3,j+z}, w_{4,j+z}, w_{o,j+z}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z-3}{5}) + 2, i)$
  - $\phi_2(w_{4,j+z-1}, w_{3,j+z}, w_{4,j+z}, w_{3,j+z+1}) = 0$, where $(u'_i, v'_i) = w(B, 3 \cdot (\frac{z-3}{5}) + 2, i)$
- For $j + z$, $z \equiv 4 \mod 5$:
  - $w_{o,j+z} = w_{4,j+z-2} \cdot 2^2 + w_{3,j+z-3} \cdot 2 + w_{4,j+z-3} + w_{o,j+z-2} \cdot 2^3$
  - $\phi_3(w_{1,j+z-2}, w_{2,j+z}, w_{1,j+z-1}, w_{2,j+z-1}, w_{4,j+z+1}, w_{o,j+z+2}) = 0$
  - $\phi_4(w_{1,j+z-2}, w_{2,j+z}, w_{1,j+z-1}, w_{2,j+z-1}, w_{4,j+z+1}, w_{o,j+z+2}) = 0$

Variable-base scalar multiplication circuit :

	w_1	w_2	w_3	w_4	w_o
j + 0	b_{n-1}	x_2	y_2	b_{n-2}	acc
j + 1	x_3	y_3	x_4	b_{n-3}	acc
j + 2	x_1	y_1	y_4	b_{n-4}	acc
j + 3	x_5	y_5	x_6	b_{n-5}	acc
j + 4	y_6	x_7	y_7	b_{n-6}	acc
...					
j + 210	...	x_{n-3}	y_{n-3}	b_2	b
j + 211	x_{n-2}	y_{n-2}	b_1	b_0	x_{n-1}
j + 212	x_1	y_1	y_{n-1}	x_n	y_n

Define the following functions:

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

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

Constraints:

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

2.5.8 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.