

# In-EVM Solana State Verification

## Technical Reference

Alisa Cherniaeva

[a.cherniaeva@nil.foundation](mailto:a.cherniaeva@nil.foundation)

=nil; Crypto3 (<https://crypto3.nil.foundation>)

Ilia Shirobokov

[i.shirobokov@nil.foundation](mailto:i.shirobokov@nil.foundation)

=nil; Crypto3 (<https://crypto3.nil.foundation>)

Mikhail Komarov

[nemo@nil.foundation](mailto:nemo@nil.foundation)

=nil; Foundation (<https://nil.foundation>)

June 20, 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Overview . . . . .	2
<b>2</b>	<b>State Proof Generator</b>	<b>3</b>
2.1	'Light-Client' State . . . . .	3
2.2	Transaction Proof . . . . .	4
2.3	Proof System . . . . .	4
2.4	Optimizations . . . . .	4
2.4.1	Batched FRI . . . . .	5
2.4.2	Hash By Column . . . . .	5
2.4.3	Hash By Subset . . . . .	5
2.5	RedShift Protocol . . . . .	5
2.5.1	Prover View . . . . .	5
2.5.2	Verifier View . . . . .	7
2.6	Circuit Definition . . . . .	8
2.6.1	Verification Circuit Overview . . . . .	8
2.6.2	SHA-256 Circuit . . . . .	9
2.6.3	SHA-512 Circuit . . . . .	12
2.6.4	Poseidon Circuit . . . . .	15
2.6.5	Merkle Tree Circuit . . . . .	16
2.6.6	Ed25519 Circuit . . . . .	16
2.6.7	Elliptic Curves Arithmetics . . . . .	17
2.6.8	Redshift Verification . . . . .	22
2.6.9	Validator Set Proof Circuit . . . . .	23
<b>3</b>	<b>In-EVM State Proof Verifier</b>	<b>24</b>
3.1	Verification Logic Architecture . . . . .	24
3.1.1	State Proof Sequence Maintenance . . . . .	25
3.2	Verification Logic API Reference . . . . .	25
3.3	Input Data Structures . . . . .	26
3.3.1	Placeholder Proof Structure . . . . .	26
3.3.2	Verification Parameters . . . . .	27
	<b>Bibliography</b>	<b>27</b>

# 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 verifier.

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.

This part's crucial components are defined by Solana's replication protocol design and consist of:

1. Input data format ('Light-Client' state data structure).
2. Transaction auxiliary proof.
3. Proof system used for the proof generation.
4. 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
- $B_k = H(B_{k-1} || \text{account\_hash} || \text{signature\_count\_buf} || b_k || \text{validators\_state})$  - bank hash of the block<sup>1</sup>
- $b_k$  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
- $\{\bar{B}_{n_1}, \dots, \bar{B}_{n_2}, \dots, \bar{B}_{n_2+32}\}$  - block information for blocks from  $n_1$  to  $n_2 + 32$ .
- $\sigma_0, \dots, \sigma_N$  - 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;
```

<sup>1</sup>See <https://docs.solana.com/proposals/simple-payment-and-state-verification#block-headers>

```

typedef SignatureSchemeType signature_scheme_type;
typedef typename signature_scheme_type::signature_type signature_type;

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 Transaction Proof

Since state proof sequence is represented as a short Merkle-tree fingerprint and does not allow to check transactions without additional mechanisms (as described in ??), there is a need to introduce additional transaction proof as simple Merkle-tree inclusion proofs with low verification costs.

Recall the current state representation on the Ethereum side.  $H(T_{n_1, n_2})$  is a Merkle tree root of blocks  $\{n_1, \dots, n_2\}$ . Each block  $n_i$  contains transactions tree  $Tx_{n_i}$  with a root  $H(Tx_{n_i})$ .

The prover wants to show that transaction  $tx$  was included in the block  $n_i$ . Let  $D_{n_1, n_2}$  be a hash table.

1. If transaction can be spent only once (i.e. it is not a "state-check" transaction):
  - 1.1 Check that  $D_{n_1, n_2}$  does not contain  $H(tx)$ .
2. The prover provides path of  $tx$  to the  $H(Tx_{n_i})$  and shows that  $H(Tx_{n_i})$  is included in  $n_i$ .
3. The prover provides the path of  $H(n_i)$  to the  $H(T_{n_1, n_2})$ .
4. If transaction can be spent only once:
  - 4.1 Add  $H(tx)$  to  $D_{n_1, n_2}$ .

This is enough to prove that the transaction was included in the confirmed state.

*Remark.* Transaction proof can be included in the original state-proof circuit. In this case, a state prover can include as many transaction proofs as they desire. It (almost) does not influence the resulting verification costs and proof size.

The transaction proof contains  $\log(n_2 - n_1) + \log(k)$  hashes where  $k$  is number of transactions in the block. Note that only the transaction's hash value will be written to Ethereum's storage. All required information is available to the light client.

## 2.3 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].

## 2.4 Optimizations

### WIP

### 2.4.1 Batched FRI

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

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

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

### 2.4.2 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.

### 2.4.3 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.

## 2.5 RedShift Protocol

### WIP

Notations:

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

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

### Preprocessing:

#### 2.5.1 Prover View

1. Choose masking polynomials:

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

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

2. Define new witness polynomials:

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

- 
1.  $\mathcal{L}' = (\mathbf{q}_0, \dots, \mathbf{q}_{N_{\text{sel}}})$
  2. Let  $\omega$  be a  $2^k$  root of unity
  3. Let  $\delta$  be a  $T$  root of unity, where  $T \cdot 2^S + 1 = p$  with  $T$  odd and  $k \leq S$
  4. Compute  $N_{\text{perm}}$  permutation polynomials  $S_{\sigma_i}(X)$  such that  $S_{\sigma_i}(\omega^j) = \delta^{i'} \cdot \omega^{j'}$
  5. Compute  $N_{\text{perm}}$  identity permutation polynomials:  $S_{id_i}(X)$  such that  $S_{id_i}(\omega^j) = \delta^i \cdot \omega^j$
  6. Let  $H = \{\omega^0, \dots, \omega^n\}$  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,  $i = 0, \dots, m$ .
- 

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

$$\begin{aligned} A(\theta) &= \theta^{m-1}A_0 + \theta^{m-2}A_1 + \dots + \theta A_{m-2} + A_{m-1} \\ S(\theta) &= \theta^{m-1}S_0 + \theta^{m-2}S_1 + \dots + \theta S_{m-2} + S_{m-1} \end{aligned}$$

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  $\text{hash}(\text{transcript})$
9. For  $0 \leq i < N_{\text{perm}}$

$$\begin{aligned} p_i &= f_i + \beta \cdot S_{id_i} + \gamma \\ q_i &= f_i + \beta \cdot S_{\sigma_i} + \gamma \end{aligned}$$

10. Define:

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

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

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

12. Compute and add commitments to  $P$  and  $Q$  to transcript
13. Compute permutation product column:

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

14. Compute and add commitments to  $V$  to transcript
15. Get  $\alpha_0, \dots, \alpha_5 \in \mathbb{F}$  from  $hash(transcript)$
16. Get  $\tau$  from  $hash(transcript)$
17. Define polynomials ( $F_0, \dots, F_4$  - copy-satisfiability,  $\mathbf{gate}_0$  is  $PI$ -constraining gate):

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

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

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

20.  $N_T := \max(N_{perm}, \mathbf{deg}_{gates} - 1)$ , where  $\mathbf{deg}_{gates}$  is the highest degree of the degrees of gate polynomials.
21. Split  $T(X)$  into separate polynomials  $T_0(X), \dots, T_{N_T-1}(X)$ <sup>2</sup>
22. Add commitments to  $T_0(X), \dots, T_{N_T-1}(X)$  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  $\pi_{comm}$  and  $\pi_{eval}$ , where:
  - $\pi_{comm} = \{f_{0,comm}, \dots, f_{N_{wires}-1,comm}, P_{comm}, Q_{comm}, T_{0,comm}, \dots, T_{N_T-1,comm}, A'_{comm}, S'_{comm}, V_{comm}\}$
  - $\pi_{eval}$  is evaluation proofs for  $f_0(y), \dots, f_{N_{wires}}(y), P(y), P(y\omega), Q(y), Q(y\omega), T_0(y), \dots, T_{N_T-1}(y), A'(y), A'(y\omega^{-1}), S'(y), V(y), V(y\omega)$

### 2.5.2 Verifier View

1. Let  $f_{0,comm}, \dots, f_{N_{wires}-1,comm}$  be commitments to  $f_0(X), \dots, f_{N_{wires}-1}(X)$
2.  $transcript = setup\_values || f_{0,comm} || \dots || f_{N_{wires}-1,comm}$
3.  $\theta = hash(transcript)$
4. Let  $A'_{comm}, S'_{comm}$  be commitments to  $A'(X), S'(X)$ .
5.  $transcript = transcript || A'_{comm} || S'_{comm}$

---

<sup>2</sup>Commit scheme supposes that polynomials should be degree  $\leq n$



6.  $\beta, \gamma = \text{hash}(\text{transcript})$
7. Let  $P_{\text{comm}}, Q_{\text{comm}}, V_{i,\text{comm}}$  be commitments to  $P(X), Q(X), V(X)$ .
8.  $\text{transcript} = \text{transcript} || P_{\text{comm}} || Q_{\text{comm}} || V_{\text{comm}}$
9.  $\alpha_0, \dots, \alpha_5 = \text{hash}(\text{transcript})$
10.  $\tau = \text{hash}(\text{transcript})$
11.  $N_T := \max(N_{\text{perm}}, \text{deg}_{\text{gates}} - 1)$ , where  $\text{deg}_{\text{gates}}$  is the highest degree of the degrees of gate polynomials.
12. Let  $T_{0,\text{comm}}, \dots, T_{N_T-1,\text{comm}}$  be commitments to  $T_0(X), \dots, T_{N_T-1}(X)$
13.  $\text{transcript} = \text{transcript} || T_{0,\text{comm}} || \dots || T_{N_T-1,\text{comm}}$
14.  $y = \text{hash}_{\mathbb{F}/H}(\text{transcript})$
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:

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

17. Check the identity:

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

## 2.6 Circuit Definition

This section contains a description of PLONK-style circuits for In-EVM Solana's "Light Client" state verification<sup>3</sup>.

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

### 2.6.1 Verification Circuit Overview

Let bank-hashes of proving block set be  $\{H_{B_{n_1}}, \dots, H_{B_{n_2}}\}$ . 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

---

<sup>3</sup><https://blog.nil.foundation/2021/10/14/solana-ethereum-bridge.html>

2. Validator set constraints. // see Section ??

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

3.1  $H_{B_i} = \text{sha256}(\text{block\_data} || H_{B_{i-1}})$  // see Section 2.6.2

4. for  $j$  from 0 to  $M$ :

4.1 Ed25519 constraints for  $H_{B_{n_2+32}}$  // see Section 2.6.6

5. Merkle tree constraints for the set  $\{H_{B_{n_1}}, \dots, H_{B_{n_2}}\}$  // see Section 2.6.5

Suppose that  $M = 800$  and  $n_2 - n_1 = 3600$ . Thus, the total amount of rows is:  $3 \cdot 3632 \cdot 755 + 800 \cdot 64432 + 3600 \cdot 22 = 59851280$

## 2.6.2 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. **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  $\sigma_0, \sigma_1$ :

$$W_i = \sigma_1(W_{i-2}) \oplus W_{i-7} \oplus \sigma_0(W_{i-15}) \oplus W_{i-16} \quad (2.1)$$

Each round of the message scheduling has the following table:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$a$	$a_0$	$a_1$	$a_2$	$a_3$	$\hat{a}_1$	$\hat{a}_2$	$a'_0$	
$j + 1$	$W_i$	$W_j$	$a'_1$	$a'_2$	$a'_3$	$s'_0$	$s'_1$	$s'_2$	$s'_3$
$j + 2$	$w$	$s_0$	$s_1$	$s_2$	$s_3$	$s_0$	$s_1$	$s_2$	$s_3$
$j + 3$		$b'_0$	$b'_1$	$b'_2$	$b'_3$	$s'_0$	$s'_1$	$s'_2$	$s'_3$
$j + 4$	$b$	$b_0$	$b_1$	$b_2$	$b_3$	$\hat{b}_0$	$\hat{b}_1$	$\hat{b}_3$	

Evaluations:

Let  $b$  be  $W_{i-2}$  and  $a$  be  $W_{i-15}$  from 2.1. The values  $W_i$  and  $W_j$  in the table corresponds to  $W_{i-7}$  and  $W_{i-16}$  respectively from 2.1. From the round  $r = 2$  the copy constraints are used for values  $b$  and  $w$  from round  $r - 2$ . The copy constraints for  $W_{i-7}, W_{i-15}$  and  $W_{i-16}$  are used in a similar way. The output of round  $W_i$  from 2.1 is  $w$ .

The first 16 words require a range check. We get it for free from range-constraining chunks inside functions  $\sigma_0$  and  $\sigma_1$ . Thus, for  $i$  from 16 to 63:

1. Apply  $\sigma_0$  to  $W_{i-15}$ .

2. Add the following constraint for  $W_i$ :

$$w_{0,j+2} = w_{0,j+1} + w_{1,j+1} + w_{1,j+2} + w_{2,j+2} \cdot 2^3 + w_{3,j+2} \cdot 2^7 + w_{4,j+2} \cdot 2^{18} + w_{5,j+2} + w_{6,j+2} \cdot 2^{10} + w_{7,j+2} \cdot 2^{17} + w_{8,j+2} \cdot 2^{19},$$

3. Apply  $\sigma_1$  to  $W_{i-2}$ .

Thus, the message schedule takes  $5 \cdot 48 = 240$  rows.

**The function  $\sigma_0$**  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  $j + \text{len}(a_i) = 14$  and  $\text{len}(a_i)$  is bit-length of  $a_i$ . The tuple  $\{s'_0, s'_1, s'_2, s'_3\}$  is a sparse representation of the result of  $\sigma_0$  and the tuple  $\{s_0, s_1, s_2, s_3\}$  is a normal representation. The size of elements of these tuples equals to  $\{14, 14, 2, 2\}$  bits respectively.

Constraints:

$$\begin{aligned} w_{0,j+0} &= w_{1,j+0} + w_{2,j+0} \cdot 2^3 + w_{3,j+0} \cdot 2^7 + w_{4,j+0} \cdot 2^{18} \\ &\quad (w_{1,j+0} - 7) \cdot (w_{1,j+0} - 6) \cdot \dots \cdot w_{1,j+0} = 0 \\ w_{5,j+1} + w_{6,j+1} \cdot 4^{14} + w_{7,j+1} \cdot 4^{28} + w_{8,j+1} \cdot 2^{30} &= w_{2,j+1} + w_{3,j+1} \cdot 4^4 + w_{4,j+1} \cdot 4^{15} + w_{3,j+1} + w_{4,j+1} \cdot \\ &\quad 4^{11} + w_{7,j+0} \cdot 4^{25} + w_{2,j+1} \cdot 4^{28} + w_{4,j+1} + w_{7,j+0} \cdot 4^{14} + w_{2,j+1} \cdot 4^{17} + w_{3,j+1} \cdot 4^{21} \\ (w_{7,j+1} - 3) \cdot (w_{7,j+1} - 2) \cdot (w_{7,j+1} - 1) \cdot w_{7,j+1} &= 0 \quad (w_{8,j+1} - 3) \cdot (w_{8,j+1} - 2) \cdot (w_{8,j+1} - 1) \cdot w_{8,j+1} = 0 \\ 10 \text{ plookup constraints: } (w_{1,j+0}, w_{7,j+0}), (2^{10} \cdot w_{2,j+0}, w_{5,j+0}), (w_{2,j+0}, w_{2,j+1}), (2^3 \cdot \\ w_{3,j+0}, w_{6,j+0}), (w_{3,j+0}, w_{3,j+1}), (w_{4,j+0}, w_{4,j+1}), (w_{1,j+2}, w_{5,j+1}), (w_{2,j+2}, w_{6,j+1}), (w_{3,j+2}, w_{7,j+2}), (w_{4,j+2}, w_{8,j+2}) \end{aligned}$$

**The function  $\sigma_1$**  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  $\sigma_0$ . The tuple  $\{s'_0, s'_1, s'_2, s'_3\}$  is a sparse representation of the result of  $\sigma_1$  and the tuple  $\{s_0, s_1, s_2, s_3\}$  is a normal representation. The size of elements of these tuples equals to  $\{14, 14, 2, 2\}$  bits respectively.

Constraints:

$$\begin{aligned} w_{0,j+3} &= w_{1,j+3} + w_{2,j+3} \cdot 2^{10} + w_{3,j+3} \cdot 2^{17} + w_{4,j+3} \cdot 2^{19} \\ &\quad (w_{3,j+3} - 3) \cdot (w_{3,j+3} - 2) \cdot (w_{3,j+3} - 1) \cdot w_{3,j+3} = 0 \\ w_{5,j+3} + w_{6,j+3} \cdot 4^{14} + w_{7,j+3} \cdot 4^{28} + w_{8,j+3} \cdot 2^{30} &= w_{2,j+3} + w_{3,j+3} \cdot 4^7 + w_{4,j+3} \cdot 4^9 + w_{3,j+3} + w_{4,j+3} \cdot \\ &\quad 4^2 + w_{1,j+3} \cdot 4^{15} + w_{2,j+3} \cdot 4^{25} + w_{4,j+3} + w_{1,j+3} \cdot 4^{13} + w_{2,j+3} \cdot 4^{23} + w_{3,j+3} \cdot 4^{30} \\ (w_{7,j+3} - 3) \cdot (w_{7,j+3} - 2) \cdot (w_{7,j+3} - 1) \cdot w_{7,j+3} &= 0 \quad (w_{8,j+3} - 3) \cdot (w_{8,j+3} - 2) \cdot (w_{8,j+3} - 1) \cdot w_{8,j+3} = 0 \\ 11 \text{ plookup constraints: } (2^4 \cdot (w_{1,j+3}, w_{5,j+3}), (2^7 \cdot w_{2,j+3}, w_{6,j+3}), (2 \cdot \\ w_{4,j+3}, w_{7,j+3}), (w_{1,j+3}, w_{1,j+2}), (w_{2,j+3}, w_{2,j+2}), (w_{3,j+3}, w_{3,j+2}), (w_{4,j+3}, w_{4,j+2}), (w_{5,j+2}, w_{5,j+3}), (w_{6,j+2}, w_{6,j+3}), (w_{7,j+2}, w_{7,j+3}), (w_{8,j+2}, w_{8,j+3})) \end{aligned}$$

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

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$e$	$e'_0$	$e_0$	$e_1$	$e_2$	$e_3$	$\hat{e}_1$	$\hat{e}_2$	$\hat{e}_3$
$j + 1$	$e'$	$f'$	$e'_1$	$e'_2$	$e'_3$	$s'_0$	$s'_1$	$s'_2$	$s'_3$
$j + 2$	$ch_{0,sparse}$	$ch_{1,sparse}$	$ch_{2,sparse}$	$ch_{3,sparse}$	—	$s_0$	$s_1$	$s_2$	$s_3$
$j + 3$	$g'$	$d$	$h$	$W_r$	$e_{new}$	$ch_0$	$ch_1$	$ch_2$	$ch_3$
$j + 4$	$maj_{0,sparse}$	$maj_{1,sparse}$	$maj_{2,sparse}$	$maj_{3,sparse}$	$a_{new}$	$maj_3$	$maj_0$	$maj_1$	$maj_2$
$j + 5$	$a'$	$b'$			$c'$	$s_0$	$s_1$	$s_2$	$s_3$
$j + 6$	$s'_1$	$s'_2$	$a'_0$	$a'_1$	$a'_2$	$a'_3$	$s'_3$	$s'_4$	
$j + 7$	$a$		$a_0$	$a_1$	$a_2$	$a_3$	$\hat{a}_0$	$\hat{a}_1$	$\hat{a}_3$



**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  $\sigma_0$ . The value of the *maj* function is stored in chunks of 8 bits  $\{maj_0, maj_1, maj_2, maj_3\}$  and the corresponded sparse value is  $\{maj_{0,sparse}, maj_{1,sparse}, maj_{2,sparse}, maj_{3,sparse}\}$  Constraints:

$$w_{0,j+4} + w_{1,j+4} \cdot 4^8 + w_{2,j+4} \cdot 4^{8 \cdot 2} + w_{3,j+4} \cdot 4^{8 \cdot 3} = w_{0,j+5} + w_{1,j+5} + w_{4,j+5}$$

4 plookup constraints:  $(w_{5,j+4}, w_{0,j+4}), (w_{6,j+4}, w_{1,j+4}), (w_{7,j+4}, w_{2,j+4}), (w_{8,j+4}, w_{3,j+4})$

**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  $\sigma_0$ . The value of the *ch* function is stored in chunks of 8 bits  $\{ch_0, ch_1, ch_2, ch_3\}$  and the corresponded sparse value is  $\{ch_{0,sparse}, ch_{1,sparse}, ch_{2,sparse}, ch_{3,sparse}\}$  Constraints:

$$w_{0,j+2} + w_{1,j+2} \cdot 7^8 + w_{2,j+2} \cdot 7^{8 \cdot 2} + w_{3,j+2} \cdot 7^{8 \cdot 3} = w_{0,j+1} + 2 \cdot w_{1,j+1} + 3 \cdot w_{0,j+3}$$

4 plookup constraints:  $(w_{5,j+3}, w_{0,j+2}), (w_{6,j+3}, w_{1,j+2}), (w_{7,j+3}, w_{2,j+2}), (w_{8,j+3}, w_{3,j+2})$

**Update the values  $a$  and  $e$**  The value  $W_r$  is a word, where  $r$  is a number of round. It has to be copy-constrained with the word  $W_r$  in the message scheduling. Constraints:

$$w_{4,j+3} = w_{1,j+3} + w_{2,j+3} + w_{5,j+2} + w_{6,j+2} \cdot 2^{14} + w_{7,j+2} \cdot 2^{28} + w_{8,j+2} \cdot 2^{30} + w_{5,j+3} + w_{6,j+3} \cdot 2^8 + w_{7,j+3} \cdot 2^{8 \cdot 2} + w_{8,j+3} \cdot 2^{8 \cdot 3} + k[r] + w_{3,j+3}, \text{ where } r \text{ is a number of round.}$$

$$w_{4,j+4} = w_{4,j+3} - w_{1,j+3} + w_{5,j+5} + w_{6,j+5} \cdot 2^{14} + w_{7,j+5} \cdot 2^{28} + w_{8,j+5} \cdot 2^{30} + w_{5,j+4} + w_{6,j+4} \cdot 2^8 + w_{7,j+4} \cdot 2^{8 \cdot 2} + w_{8,j+4} \cdot 2^{8 \cdot 3}$$

**Output of the round**

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$\bar{a}$	$\bar{b}$	$\bar{c}$	$\bar{d}$	$\bar{e}$	$\bar{f}$	—	—	—
$j + 1$	$h_0$	$h_1$	$h_2$	$h_3$	$h_4$	$h_5$	—	—	—
$j + 2$	$a$	$b$	$c$	$d$	$e$	$f$	—	—	—
$j + 3$	$h_6$	$h_7$	$\bar{g}$	$\bar{h}$	$g$	$h$	—	—	—

Evaluations:

The values  $\bar{\xi}$  copy constrained with initial working variables of this round. The values  $a, b, c, d, e, f, g, h$  copy constrained with variables from the compression. The output of the round is  $h_0, h_1, \dots, h_7$

Constraints:

$$\begin{aligned} w_{0,j+1} &= w_{0,j+0} + w_{0,j+2} \\ w_{1,j+1} &= w_{1,j+0} + w_{1,j+2} \\ w_{2,j+1} &= w_{2,j+0} + w_{2,j+2} \\ w_{3,j+1} &= w_{3,j+0} + w_{3,j+2} \\ w_{4,j+1} &= w_{4,j+0} + w_{4,j+2} \\ w_{5,j+1} &= w_{5,j+0} + w_{5,j+2} \\ w_{0,j+3} &= w_{2,j+3} + w_{4,j+3} \\ w_{1,j+3} &= w_{3,j+3} + w_{5,j+3} \end{aligned}$$

**Cost** The total value of rows is  $48 \cdot 5 + 8 \cdot 64 + 3 = 755$  per chunk.

### 2.6.3 SHA-512 Circuit

SHA-512 uses the similar logical functions as in 2.6.2 which operates on 64-bits words. Thus, the preprocessing constraints for the SHA-512‘ circuit is a decomposition of  $k$  message blocks to 64 bits chunks without range proofs. For ‘eddsa‘ circuit,  $k = 2$ . All evaluations are similar to SHA-256 circuit.

**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. **SHA-512 NORMALIZE MAJ** with 2 columns and  $2^{16}$  rows. The first column contains all possible 16-bits words. The second column contains corresponding sparse representations with base 4.
4. **SHA-512 NORMALIZE CH** with 2 columns and  $2^{16}$  rows. The first column contains all possible 16-bits words. The second column contains corresponding sparse representations with base 7.

**Message scheduling** For each block of 1024 bits of the padded message the 80 words are constructed in the following way:

- The first 16 words are obtained by splitting the message.
- The last 64 words are obtained by using the functions  $\sigma_0, \sigma_1$ :

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

Each round of the message scheduling has the following table:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$a$	$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$\hat{a}_1$
$j + 1$		$a'_0$	$a'_1$	$a'_2$	$a'_3$	$a'_4$	$a'_5$	$a'_6$	
$j + 2$	$W_i$	$s'_0$	$s'_1$	$s'_2$	$s'_3$	$s'_4$	$\hat{s}'_4$	--	$W_j$
$j + 3$	$w$	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$			
$j + 4$	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$	$s'_2$	$s'_3$	$s'_4$	$\hat{s}'_4$
$j + 5$	$s'_0$	$b'_0$	$b'_1$	$b'_2$	$b'_3$	$b'_4$	$b'_5$	$s'_1$	--
$j + 6$	$b$	$b_0$	$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$\hat{b}_0$	$\hat{b}_5$

The first 16 words require a range check. We get it for free from range-constraining chunks inside functions  $\sigma_0$  and  $\sigma_1$ . Thus, for  $i$  from 16 to 80:

1. Apply  $\sigma_0$  to  $W_{i-15}$ .
2. Add the following constraint for  $W_i$ :

$$w_{0,j+3} = w_{0,j+2} + w_{8,j+2} + w_{1,j+3} + w_{2,j+3} \cdot 2^{14} + w_{3,j+3} \cdot 2^{28} + w_{4,j+3} \cdot 2^{42} + w_{5,j+3} \cdot 2^{56} + w_{0,j+4} + w_{1,j+4} \cdot 2^{14} + w_{2,j+4} \cdot 2^{28} + w_{3,j+4} \cdot 2^{42} + w_{4,j+4} \cdot 2^{56},$$

3. Apply  $\sigma_1$  to  $W_{i-2}$ .

Thus, the message schedule takes  $7 \cdot 64 = 448$  rows.

**The function  $\sigma_0$**  contains sparse mapping with base 4. Let  $a$  be divided to chunks  $a_0, a_1, a_2, a_3, a_4, a_5, a_6$  which equals to 1, 6, 1, 14, 14, 14, 14 bits respectively. The values  $a'_0, a'_1, a'_2, a'_3, a'_4, a'_5, a'_6$  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  $j + \text{len}(a_i) = 14$  and  $\text{len}(a_i)$  is bit-length of  $a_i$ .

Constraints:

$$\begin{aligned}
w_{0,j+0} &= w_{1,j+0} + w_{2,j+0} \cdot 2 + w_{3,j+0} \cdot 2^7 + w_{4,j+0} \cdot 2^8 + w_{5,j+0} \cdot 2^{22} + w_{6,j+0} \cdot 2^{36} + w_{7,j+0} \cdot 2^{50} \\
&\quad (w_{1,j+0} - 1) \cdot w_{1,j+0} = 0 \\
&\quad (w_{3,j+0} - 1) \cdot w_{3,j+0} = 0 \\
w_{1,j+2} + w_{2,j+2} \cdot 4^{14} + w_{3,j+2} \cdot 4^{28} + w_{4,j+2} \cdot 2^{42} + w_{5,j+2} \cdot 4^{56} &= w_{2,j+1} + w_{3,j+1} \cdot 4^6 + w_{4,j+1} \cdot 4^7 + \\
w_{5,j+1} \cdot 2^{21} + w_{6,j+1} \cdot 4^{35} + w_{7,j+1} \cdot 4^{49} + w_{1,j+1} \cdot 4^{63} + w_{3,j+1} + w_{4,j+1} \cdot 4 + w_{5,j+1} \cdot 4^{15} + w_{6,j+1} \cdot 2^{29} + \\
w_{7,j+1} \cdot 4^{43} + w_{4,j+1} + w_{5,j+1} \cdot 4^{14} + w_{6,j+1} \cdot 4^{28} + w_{7,j+1} \cdot 2^{42} + w_{1,j+1} \cdot 4^{56} + w_{2,j+1} \cdot 4^{57} + w_{3,j+1} \cdot 4^{63} \\
&\quad 15 \text{ plookup constraints: } (w_{1,j+0}, w_{1,j+1}), (2^8 \cdot \\
w_{2,j+0}, w_{8,j+0}), (w_{2,j+0}, w_{2,j+1}), (w_{3,j+0}, w_{3,j+1}), (w_{4,j+0}, w_{4,j+1}), (w_{5,j+0}, w_{5,j+1}), (w_{6,j+0}, w_{6,j+1}), (w_{7,j+0}, w_{7,j+1}), (w_{5,j+3}, w_{6,j+2})
\end{aligned}$$

**The function  $\sigma_1$**  contains sparse mapping subcircuit with base 4. Let  $a$  be divided to chunks  $a_0, a_1, a_2, a_3, a_4, a_5$  which equals to 6, 13, 14, 14, 14, 3 bits respectively. The values  $a'_0, a'_1, a'_2, a'_3, a'_4, a'_5$  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  $\sigma_0$ .

Constraints:

$$\begin{aligned}
w_{0,j+6} &= w_{1,j+6} + w_{2,j+6} \cdot 2^6 + w_{3,j+6} \cdot 2^{19} + w_{4,j+6} \cdot 2^{33} + w_{5,j+6} \cdot 2^{47} + w_{6,j+6} \cdot 2^{61} \\
&\quad (w_{6,j+6} - 7) \cdot (w_{6,j+6} - 6) \cdot \dots \cdot w_{6,j+6} = 0 \\
&\quad w_{0,j+5} + w_{7,j+5} \cdot 4^{14} + w_{5,j+4} \cdot 4^{28} + w_{6,j+4} \cdot 2^{42} + w_{7,j+4} \cdot 4^{56} = \\
&\quad w_{2,j+5} + w_{3,j+5} \cdot 4^{13} + w_{4,j+5} \cdot 4^{27} + w_{5,j+5} \cdot 2^{41} + w_{6,j+5} \cdot 4^{55} + w_{3,j+5} + w_{4,j+5} \cdot 4^{14} + w_{5,j+5} \cdot 4^{28} + w_{6,j+5} \cdot \\
&\quad 2^{42} + w_{1,j+5} \cdot 4^{45} + w_{2,j+5} \cdot 4^{51} + w_{6,j+5} + w_{1,j+5} \cdot 4^3 + w_{2,j+5} \cdot 4^9 + w_{3,j+5} \cdot 2^{22} + w_{4,j+5} \cdot 4^{36} + w_{5,j+5} \cdot 4^{50} \\
&\quad 15 \text{ plookup constraints: } (w_{1,j+6}, w_{1,j+5}), (2^8 \cdot \\
&\quad w_{1,j+6}, w_{7,j+6}), (w_{2,j+6}, w_{2,j+5}), (w_{3,j+6}, w_{3,j+5}), (w_{4,j+6}, w_{4,j+5}), (w_{5,j+6}, w_{5,j+5}), (w_{6,j+6}, w_{6,j+5}), (2 \cdot \\
&\quad w_{6,j+6}, w_{8,j+6}), (w_{0,j+4}, w_{0,j+5}), (w_{1,j+4}, w_{7,j+5}), (w_{2,j+4}, w_{5,j+4}), (w_{3,j+4}, w_{6,j+4}), (w_{4,j+4}, w_{7,j+4}), (2^6 \cdot \\
&\quad w_{4,j+4}, w_{8,j+4})
\end{aligned}$$

**Compression** There are 80 rounds of compression. Each round of compression has the following table:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$e$	$e_0$	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$\hat{e}_1$	$\hat{e}_3$
$j + 1$	$--$	$e'_0$	$e'_1$	$e'_2$	$e'_3$	$e'_4$	$e'_5$	$\hat{e}_5$	$--$
$j + 2$	$e'$	$f'$	$--$	$\hat{s}'_4$	$s'_0$	$s'_1$	$s'_2$	$s'_3$	$s'_4$
$j + 3$	$ch_{0,sparse}$	$ch_{1,sparse}$	$ch_{2,sparse}$	$ch_{3,sparse}$	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$j + 4$	$g'$	$--$	$--$	$--$	$e_{new}$	$ch_0$	$ch_1$	$ch_2$	$ch_3$
$j + 5$	$c'$	$d$	$h$	$W_r$	$a_{new}$	$maj_3$	$maj_0$	$maj_1$	$maj_2$
$j + 6$	$maj_{0,sparse}$	$maj_{1,sparse}$	$maj_{2,sparse}$	$maj_{3,sparse}$	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$j + 7$	$a'$	$b'$	$--$	$\hat{s}'_4$	$s'_0$	$s'_1$	$s'_2$	$s'_3$	$s'_4$
$j + 8$		$a'_0$	$a'_1$	$a'_2$	$a'_3$	$a'_4$	$a'_5$	$\hat{a}_5$	$--$
$j + 9$	$a$	$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$\hat{a}_2$	$\hat{a}_3$

**The working variables**  $a, b, c, d, e, f, g, h$  equals to the fixed initial  $SHA - 512$  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:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$a$	$a'$	$b$	$b'$	$d$	$--$	$--$	$--$	$--$
$j + 1$	$c$	$c'$	$e$	$e'$	$h$	$--$	$--$	$--$	$--$
$j + 2$	$f$	$f'$	$g$	$g'$	$--$	$--$	$--$	$--$	$--$

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  $\Sigma_0$  function** contains subcircuit with base 4. Let  $a$  be divided to chunks  $a_0, a_1, a_2, a_3, a_4, a_5$  which equals to 14, 14, 6, 5, 14, 11 bits respectively. The values  $a'_0, a'_1, a'_2, a'_3, a'_4, a'_5$  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  $\sigma_0$ .

Constraints:

$$\begin{aligned}
w_{0,j+9} &= w_{1,j+9} + w_{2,j+9} \cdot 2^{14} + w_{3,j+9} \cdot 2^{28} + w_{4,j+9} \cdot 2^{34} + w_{5,j+9} \cdot 2^{39} + w_{6,j+9} \cdot 2^{53} \\
&\quad w_{4,j+7} + w_{5,j+7} \cdot 4^{14} + w_{6,j+7} \cdot 4^{28} + w_{7,j+7} \cdot 2^{42} + w_{8,j+7} \cdot 4^{56} = \\
&\quad w_{3,j+8} + w_{4,j+8} \cdot 4^6 + w_{5,j+8} \cdot 4^{11} + w_{6,j+8} \cdot 2^{25} + w_{1,j+8} \cdot 4^{36} + w_{2,j+8} \cdot 4^{50} + w_{4,j+8} + w_{5,j+8} \cdot 4^5 + w_{6,j+8} \cdot 4^{19} + \\
&\quad w_{1,j+8} \cdot 2^{30} + w_{2,j+8} \cdot 4^{44} + w_{3,j+8} \cdot 4^{58} + w_{5,j+8} + w_{6,j+8} \cdot 4^{14} + w_{1,j+8} \cdot 4^{25} + w_{2,j+8} \cdot 2^{39} + w_{3,j+8} \cdot 4^{53} + w_{4,j+8} \cdot 4^{59} \\
&\quad 15 \text{ plookup constraints: } (w_{1,j+9}, w_{1,j+8}), (w_{2,j+9}, w_{2,j+8}), (2^8 \cdot w_{3,j+9}, w_{7,j+9}), (w_{3,j+9}, w_{3,j+8}), (2^9 \cdot \\
&\quad w_{4,j+9}, w_{8,j+9}), (w_{4,j+9}, w_{4,j+8}), (w_{5,j+9}, w_{5,j+8}), (2^3 \cdot \\
&\quad w_{6,j+9}, w_{7,j+8}), (w_{6,j+9}, w_{6,j+8}), (w_{4,j+6}, w_{4,j+7}), (w_{5,j+6}, w_{5,j+7}), (w_{6,j+6}, w_{6,j+7}), (w_{7,j+6}, w_{7,j+7}), (w_{8,j+6}, w_{8,j+7}), (2^{10} \cdot \\
&\quad w_{8,j+7}, w_{3,j+7})
\end{aligned}$$

**The  $\Sigma_1$  function** contains subcircuit with base 7. Let  $a$  be divided to chunks  $a_0, a_1, a_2, a_3, a_4, a_5$  which equals to 14, 4, 14, 9, 14, 9 bits respectively. The values  $a'_0, a'_1, a'_2, a'_3, a'_4, a'_5$  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  $\sigma_0$ .

Constraints:

$$\begin{aligned}
w_{0,j+0} &= w_{1,j+0} + w_{2,j+0} \cdot 2^{14} + w_{3,j+0} \cdot 2^{18} + w_{4,j+0} \cdot 2^{32} + w_{5,j+0} \cdot 2^{41} + w_{6,j+0} \cdot 2^{55} \\
&\quad w_{4,j+2} + w_{5,j+2} \cdot 4^{14} + w_{6,j+2} \cdot 4^{28} + w_{7,j+2} \cdot 2^{42} + w_{8,j+2} \cdot 4^{56} = \\
&\quad w_{2,j+1} + w_{3,j+1} \cdot 4^4 + w_{4,j+1} \cdot 4^{18} + w_{5,j+1} \cdot 2^{27} + w_{6,j+1} \cdot 4^{41} + w_{1,j+1} \cdot 4^{50} + w_{3,j+1} + w_{4,j+1} \cdot 4^{14} + w_{5,j+1} \cdot 4^{23} + \\
&\quad w_{6,j+1} \cdot 2^{37} + w_{1,j+1} \cdot 4^{46} + w_{3,j+1} \cdot 4^{60} + w_{5,j+1} + w_{6,j+1} \cdot 4^{14} + w_{1,j+1} \cdot 4^{23} + w_{2,j+1} \cdot 2^{37} + w_{3,j+1} \cdot 4^{41} + w_{4,j+1} \cdot 4^{55} \\
&\quad 15 \text{ plookup constraints: } (w_{1,j+0}, w_{1,j+1}), (w_{2,j+0}, w_{2,j+1}), (2^{10} \cdot w_{2,j+0}, w_{7,j+0}), (w_{3,j+0}, w_{3,j+1}), (2^5 \cdot \\
&\quad w_{4,j+0}, w_{8,j+0}), (w_{4,j+0}, w_{4,j+1}), (w_{5,j+0}, w_{5,j+1}), (2^3 \cdot \\
&\quad w_{6,j+0}, w_{7,j+1}), (w_{6,j+0}, w_{6,j+1}), (w_{4,j+3}, w_{4,j+2}), (w_{5,j+3}, w_{5,j+2}), (w_{6,j+3}, w_{6,j+2}), (w_{7,j+3}, w_{7,j+2}), (w_{8,j+3}, w_{8,j+2}), (2^{10} \cdot \\
&\quad w_{8,j+3}, w_{3,j+2})
\end{aligned}$$

**The Maj function** contains subcircuit with base 4 for  $a, b, c$ . **SHA-512 NORMALIZE MAJ** lookup table is used for mapping to sparse representation in the same way as for  $\sigma_0$ . The value of the *maj* function is stored in chunks of 16 bits. Constraints:

$$\begin{aligned}
w_{0,j+6} + w_{1,j+6} \cdot 4^{16} + w_{2,j+6} \cdot 4^{16 \cdot 2} + w_{3,j+6} \cdot 4^{16 \cdot 3} &= w_{0,j+7} + w_{1,j+7} + w_{0,j+5} \\
4 \text{ plookup constraints: } (w_{5,j+5}, w_{0,j+6}), (w_{6,j+5}, w_{1,j+6}), (w_{7,j+5}, w_{2,j+6}), (w_{8,j+5}, w_{3,j+6})
\end{aligned}$$

**The Ch function** contain sparse mapping subcircuit with base 7 for  $e, f, g$ . **SHA-512 NORMALIZE CH** lookup table is used for mapping to sparse representation in the same way as for  $\sigma_0$ . The value of the *ch* function is stored in chunks of 16 bits. Constraints:

$$\begin{aligned}
w_{0,j+3} + w_{1,j+3} \cdot 7^{16} + w_{2,j+3} \cdot 7^{16 \cdot 2} + w_{3,j+3} \cdot 7^{16 \cdot 3} &= w_{0,j+2} + 2 \cdot w_{1,j+2} + 3 \cdot w_{0,j+4} \\
4 \text{ plookup constraints: } (w_{5,j+4}, w_{0,j+3}), (w_{6,j+4}, w_{1,j+3}), (w_{7,j+4}, w_{2,j+3}), (w_{8,j+3}, w_{3,j+2})
\end{aligned}$$

**Update the values  $a$  and  $e$**  Constraints:

$$\begin{aligned}
w_{4,j+4} &= w_{1,j+5} + w_{2,j+5} + w_{5,j+3} \cdot 2^{14} + w_{6,j+3} \cdot 2^{28} + w_{7,j+3} \cdot 2^{42} + w_{8,j+3} \cdot 2^{56} + w_{5,j+4} + w_{6,j+4} \cdot 2^{16} + \\
&\quad w_{7,j+4} \cdot 2^{16 \cdot 2} + w_{8,j+4} \cdot 2^{16 \cdot 3} + k[r] + w_{3,j+5}, \text{ where } r \text{ is a number of round.} \\
w_{4,j+5} &= w_{4,j+4} - w_{1,j+5} + w_{4,j+6} + w_{5,j+6} \cdot 2^{14} + w_{6,j+6} \cdot 2^{28} + w_{7,j+6} \cdot 2^{42} + w_{8,j+6} \cdot 2^{56} + w_{5,j+5} + \\
&\quad w_{6,j+5} \cdot 2^{16} + w_{7,j+5} \cdot 2^{16 \cdot 2} + w_{8,j+5} \cdot 2^{16 \cdot 3}
\end{aligned}$$

**Output of the round** The final calculations uses the same table and constraints as in 2.6.2.

**Cost** The total value of rows is  $64 \cdot 7 + 10 \cdot 80 + 3 = 1248$  per chunk.

## 2.6.4 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:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$0_{\mathbb{F}}$	$I[2]$	$I[3]$	$T_{1,0}$	$T_{1,1}$	$T_{1,2}$	$T_{2,0}$	$T_{2,1}$	$T_{2,2}$
$j + 1$	$T_{3,0}$	$T_{3,1}$	$T_{3,2}$	$T_{4,0}$	$T_{4,1}$	$T_{4,2}$	$T_{5,0}$	$T_{5,1}$	$T_{5,2}$
...									
$j + 21$	$T_{63,0}$	$T_{63,1}$	$T_{63,2}$	$T_{64,0}$	$T_{64,1}$	$T_{64,2}$	$O[1]$	$H$	$O[3]$



Constraints:

$$\begin{aligned}
& \text{For } j + 0: \\
& [w_{3,j+0}, w_{4,j+0}, w_{5,j+0}] = [w_{0,j+0}^5, w_{1,j+0}^5, w_{2,j+0}^5] \times M + RC \\
& [w_{6,j+0}, w_{7,j+0}, w_{8,j+0}] = [w_{2,j+0}^5, w_{3,j+0}^5, w_{4,j+0}^5] \times M + RC \\
& \text{For } j + 1: \\
& [w_{0,j+1}, w_{1,j+1}, w_{2,j+1}] = [w_{2,j+0}^5, w_{7,j+0}^5, w_{8,j+0}^5] \times M + RC \\
& [w_{3,j+1}, w_{4,j+1}, w_{5,j+1}] = [w_{0,j+1}^5, w_{1,j+1}^5, w_{2,j+1}^5] \times M + RC \\
& [w_{6,j+1}, w_{7,j+1}, w_{8,j+1}] = [w_{3,j+1}, w_{4,j+1}, w_{5,j+1}^5] \times M + RC \\
& \text{For } j + k, k \in \{2, 19\}: \\
& [w_{0,j+k}, w_{1,j+k}, w_{2,j+k}] = [w_{6,j+k-1}, w_{7,j+k-1}, w_{8,j+k-1}^5] \times M + RC \\
& [w_{3,j+k}, w_{4,j+k}, w_{5,j+k}] = [w_{0,j+k}, w_{1,j+k}, w_{2,j+k}^5] \times M + RC \\
& [w_{6,j+k}, w_{7,j+k}, w_{8,j+k}] = [w_{3,j+k}, w_{4,j+k}, w_{5,j+k}^5] \times M + RC \\
& \text{For } j + 20: \\
& [w_{0,j+20}, w_{1,j+20}, w_{2,j+20}] = [w_{6,j+19}, w_{7,j+19}, w_{8,j+19}^5] \times M + RC \\
& [w_{3,j+20}, w_{4,j+20}, w_{5,j+20}] = [w_{0,j+20}, w_{1,j+20}, w_{2,j+20}^5] \times M + RC \\
& [w_{6,j+20}, w_{7,j+20}, w_{8,j+20}] = [w_{2,j+20}^5, w_{3,j+20}^5, w_{4,j+20}^5] \times M + RC \\
& \text{For } j + 21: \\
& [w_{0,j+21}, w_{1,j+21}, w_{2,j+21}] = [w_{2,j+20}^5, w_{7,j+20}^5, w_{8,j+20}^5] \times M + RC \\
& [w_{3,j+21}, w_{4,j+21}, w_{5,j+21}] = [w_{0,j+21}^5, w_{1,j+21}^5, w_{2,j+21}^5] \times M + RC \\
& [w_{6,j+21}, w_{7,j+21}, w_{8,j+21}] = [w_{2,j+21}^5, w_{3,j+21}^5, w_{4,j+21}^5] \times M + RC
\end{aligned}$$

### 2.6.5 Merkle Tree Circuit

Merkle Tree generation for set  $\{H_{B_{n_1}}, \dots, H_{B_{n_2}}\}$ . 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  $T_i := H_i$  // 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  $T'_i = \text{hash}(T_{2 \cdot i}, T_{2 \cdot i + 1})$ . // see Section 2.6.4
  - 4.2  $n = \frac{n}{2}$
  - 4.3 for  $j$  from 0 to  $n - 1$ :
    - 4.3.1  $T_i := T'_i$ . // just notation for simplicity, not a real part of the circuit

### 2.6.6 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  $L = 2^{252} + 2774231777372353535851937790883648493$ . It costs 2 rows.
2.  $k = \{k_0, k_1, \dots, k_7\} == \text{SHA-512}(data || R || A || M)$  // See section ?? It costs  $1248 \cdot 2 = 2496$  rows.
3.  $sB = ?R + kA$ :
  - 3.1 Fixed-base scalar multiplication circuit is used for  $sB = S$ . The cell  $w_{0,j+84}$  is copy-constrained with  $w_{0,j+0}$  from the range circuit.
  - 3.2 One addition is used for  $S + (-R)$ .
  - 3.3 Variable-base scalar multiplication circuit for  $T = k \cdot A$ , where cells  $w_{1,j+254}, w_{2,j+254}$  are copy constrained with  $w_{3,j+0}, w_{5,j+0}$ .

It costs  $2 + 2496 + 10880 + 93 + 4 + 50957 = 64432$  rows.

## 2.6.7 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  $121665/121666$ .

**Computations over a non-native field.** Let  $\mathbb{F}_p$  be an edwards25519 field, i.e. the size of the field is  $2^{255} - 19$ . In order to provide computations over non-native  $\mathbb{F}_p$  we use constraints over native field  $\mathbb{F}_k$ . Let  $k < p$  be a prime number, which size is 254 bits. Additionally, we compute an integer  $t$ , such that  $2^t \cdot k \geq p^2 + p$ . In our case,  $t = 257$ . Now, we want to check equality:

$$a \cdot b = p \cdot q + r, r = a \cdot b \mod p$$

Each positive integer  $a, b, q, r$  is divided into 13 limbs, where the sizes of limbs are 20, 20, ..., 20, 15 bits respectively, where 15 is the least significant bits. To check that  $a, b, q$  and  $r$  are less than  $p$ , we use range proofs. For this purpose, a lookup table with two columns is used. The first column contains all integers in the range  $[0, 2^{20})$ , and the second column contains almost all zeros except 18 ones from  $2^{15} - 19$  to  $2^{15} - 1$ .

1. The limbs  $a_0, a_1, \dots, a_{12}$  are range-constrained by the lookup table.
2. The value  $a_{12} \cdot 2^5$  are range-constrained by the lookup table.
3. Let  $\xi = (\sum_{i=0}^{11} (a_i - 2^{20} + 1))^{-1}$ .
4.  $(\sum_{i=0}^{11} (a_i - 2^{20} + 1) \cdot (\xi \cdot (\sum_{i=0}^{11} (a_i - 2^{20} + 1) - 1) - 1) = 0$
5.  $\xi \cdot (\sum_{i=0}^{11} (a_i - 2^{20} + 1) + (1 - \xi \cdot (\sum_{i=0}^{11} (a_i - 2^{20} + 1))) \cdot c - 1 = 0$ , where  $c$  is corresponding second column's value for  $a_{12}$ .

Then we constrain the equation modulo  $n$  and  $2^t$  as follows:

1.  $(a \cdot b) \mod k = (p \cdot q + r) \mod k$
2.  $a'_0 = a_{12} + a_{11} \cdot 2^{15} + a_{10} \cdot 2^{35} + a_9 \cdot 2^{55}$ ,  $a'_1 = a_8 + a_7 \cdot 2^{20} + a_6 \cdot 2^{40}$ ,  $a'_1 = a_5 + a_4 \cdot 2^{20} + a_3 \cdot 2^{40}$ ,  $a'_1 = a_2 + a_1 \cdot 2^{20} + a_0 \cdot 2^{40}$ . The new limbs for  $b, q$ , and  $r$  are constructed similarly.
3. Let  $p'$  be  $-p \mod 2^t$  and  $p' = p'_0 + p'_1 \cdot 2^{75} + p'_2 \cdot 2^{135} + p'_3 \cdot 2^{195}$ . The limbs  $p'_0, p'_1, p'_2$  and  $p'_3$  are circuits parameters.
4. Compute the following limbs:
  - 4.1  $t_0 = a'_0 \cdot b'_0 + p'_0 \cdot q'_0$
  - 4.2  $t_1 = a'_1 \cdot b'_0 + a'_0 \cdot b'_1 + p'_0 \cdot q'_1 + p'_1 \cdot q'_0$
  - 4.3  $t_2 = a'_2 \cdot b'_0 + a'_0 \cdot b'_2 + a'_1 \cdot b'_1 + p'_0 \cdot q'_2 + p'_2 \cdot q'_0 + p'_1 \cdot q'_1$
  - 4.4  $t_3 = a'_3 \cdot b'_0 + a'_0 \cdot b'_3 + a'_1 \cdot b'_2 + a'_2 \cdot b'_1 + p'_0 \cdot q'_3 + p'_3 \cdot q'_0 + p'_1 \cdot q'_2 + p'_2 \cdot q'_1$
  - 4.5  $t_4 = a'_3 \cdot b'_1 + a'_1 \cdot b'_3 + a'_2 \cdot b'_2 + p'_1 \cdot q'_3 + p'_3 \cdot q'_1 + p'_2 \cdot q'_2$
5.  $u_0 = t_0 - r'_0 + t_1 \cdot 2^{75} - r'_1 \cdot 2^{75} = v_0 \cdot 2^{135}$
6.  $u_1 = t_2 - r'_2 + t_3 \cdot 2^{60} - r'_3 \cdot 2^{60} + t_4 \cdot 2^{120} + v_0 = v_1 \cdot 2^{122}$
7. The value  $v_0$  has to be less than  $2^{68}$  and  $v_1 \leq 2^{78}$ .
  - 7.1  $v_0 = v_{0,3} + v_{0,2} \cdot 2^8 + v_{0,1} \cdot 2^{28} + v_{0,0} \cdot 2^{48}$
  - 7.2 Lookup constraints:  $(v_{0,3}), (v_{0,2}), (v_{0,1}), (v_{0,0}), (v_{0,3} \cdot 2^{12})$
  - 7.3  $v_1 = v_{1,3} + v_{1,2} \cdot 2^{18} + v_{1,1} \cdot 2^{38} + v_{1,0} \cdot 2^{58}$
  - 7.4 Lookup constraints:  $(v_{1,3}), (v_{1,2}), (v_{1,1}), (v_{1,0}), (v_{1,3} \cdot 2^2)$

## Non-native multiplication circuit for $a \cdot b$

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$a'_0$	$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$\xi$
$j + 1$	$a'_1$	$a'_2$	$a_7$	$a_8$	$a_9$	$a_{10}$	$a_{11}$	$a_{12}$	$c$
$j + 2$	$b'_0$	$b_0$	$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$b_6$	$\xi$
$j + 3$	$b'_1$	$b'_2$	$b_7$	$b_8$	$b_9$	$b_{10}$	$b_{11}$	$b_{12}$	$c$
$j + 4$	$q'_0$	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$\xi$
$j + 5$	$q'_1$	$q'_2$	$q_7$	$q_8$	$q_9$	$q_{10}$	$q_{11}$	$q_{12}$	$c$
$j + 6$	$r'_0$	$r_0$	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$\xi$
$j + 7$	$r'_1$	$r'_3$	$r_7$	$r_8$	$r_9$	$r_{10}$	$r_{11}$	$r_{12}$	$c$
$j + 8$	$q'_0$	$q'_1$	$q'_2$	$r'_3$	$r'_2$	$r'_1$	$r'_0$	$q_0$	$q_1$
$j + 9$	$b'_1$	$b'_2$	$q_2$	$b_0$	$b_1$	$b_2$	—	$v_0$	$v_1$
$j + 10$	$a'_0$	$a'_1$	$a'_2$	$b'_0$	$a_0$	$a_1$	$a_2$	$v_{0,3}$	$v_{1,3}$
$j + 11$	$v_0$	$v_{0,0}$	$v_{0,1}$	$v_{0,2}$	$v_1$	$v_{1,0}$	$v_{1,1}$	$v_{1,2}$	—

Let  $s_a$  be  $(w_{1,j+0} + w_{2,j+0} + w_{3,j+0} + w_{4,j+0} + w_{5,j+0} + w_{6,j+0} + w_{7,j+0} + w_{2,j+1} + w_{3,j+1} + w_{4,j+1} + w_{5,j+1} + w_{6,j+1} - 12 \cdot (2^{20} - 1))$ .

Let  $s_b$  be  $(w_{1,j+2} + w_{2,j+2} + w_{3,j+2} + w_{4,j+2} + w_{5,j+2} + w_{6,j+2} + w_{7,j+2} + w_{2,j+3} + w_{3,j+3} + w_{4,j+3} + w_{5,j+3} + w_{6,j+3} - 12 \cdot (2^{20} - 1))$ .

Let  $s_q$  be  $(w_{1,j+4} + w_{2,j+4} + w_{3,j+4} + w_{4,j+4} + w_{5,j+4} + w_{6,j+4} + w_{7,j+4} + w_{2,j+5} + w_{3,j+5} + w_{4,j+5} + w_{5,j+5} + w_{6,j+5} - 12 \cdot (2^{20} - 1))$ .

Let  $s_r$  be  $(w_{1,j+6} + w_{2,j+6} + w_{3,j+6} + w_{4,j+6} + w_{5,j+6} + w_{6,j+6} + w_{7,j+6} + w_{2,j+7} + w_{3,j+7} + w_{4,j+7} + w_{5,j+7} + w_{6,j+7} - 12 \cdot (2^{20} - 1))$ .

Constraints:

- $s_a \cdot (w_{8,j+0} \cdot s_a - 1) = 0$
- $w_{8,j+0} \cdot (s_a) + (1 - w_{8,j+0} \cdot s_a) \cdot w_{8,j+1} - 1 = 0$
- $w_{0,j+0} = w_{7,j+1} + w_{6,j+1} \cdot 2^{15} + w_{5,j+1} \cdot 2^{35} + w_{4,j+1} \cdot 2^{55}$
- $w_{0,j+1} = w_{3,j+1} + w_{2,j+1} \cdot 2^{20} + w_{7,j+0} \cdot 2^{40}$
- $w_{1,j+1} = w_{6,j+0} + w_{5,j+0} \cdot 2^{20} + w_{4,j+0} \cdot 2^{40}$
- $s_b \cdot (w_{8,j+2} \cdot s_b - 1) = 0$
- $w_{8,j+2} \cdot (s_b) + (1 - w_{8,j+2} \cdot s_b) \cdot w_{8,j+3} - 1 = 0$
- $w_{0,j+2} = w_{7,j+3} + w_{6,j+3} \cdot 2^{15} + w_{5,j+3} \cdot 2^{35} + w_{4,j+3} \cdot 2^{55}$
- $w_{0,j+3} = w_{3,j+3} + w_{2,j+3} \cdot 2^{20} + w_{7,j+2} \cdot 2^{40}$
- $w_{1,j+3} = w_{6,j+2} + w_{5,j+2} \cdot 2^{20} + w_{4,j+2} \cdot 2^{40}$
- $s_q \cdot (w_{8,j+4} \cdot s_q - 1) = 0$
- $w_{8,j+4} \cdot (s_q) + (1 - w_{8,j+4} \cdot s_q) \cdot w_{8,j+5} - 1 = 0$
- $w_{0,j+4} = w_{7,j+5} + w_{6,j+5} \cdot 2^{15} + w_{5,j+5} \cdot 2^{35} + w_{4,j+5} \cdot 2^{55}$
- $w_{0,j+5} = w_{3,j+5} + w_{2,j+5} \cdot 2^{20} + w_{7,j+4} \cdot 2^{40}$
- $w_{1,j+5} = w_{6,j+4} + w_{5,j+4} \cdot 2^{20} + w_{4,j+4} \cdot 2^{40}$
- $s_r \cdot (w_{8,j+6} \cdot s_r - 1) = 0$
- $w_{8,j+6} \cdot (s_r) + (1 - w_{8,j+6} \cdot s_r) \cdot w_{8,j+7} - 1 = 0$
- $w_{4,j+8} = w_{7,j+7} + w_{6,j+7} \cdot 2^{15} + w_{5,j+7} \cdot 2^{35} + w_{4,j+7} \cdot 2^{55}$
- $w_{0,j+7} = w_{3,j+7} + w_{2,j+7} \cdot 2^{20} + w_{7,j+6} \cdot 2^{40}$
- $w_{1,j+7} = w_{6,j+6} + w_{5,j+6} \cdot 2^{20} + w_{4,j+6} \cdot 2^{40}$
- $w_{1,j+7} = w_{1,j+6} + w_{2,j+6} \cdot 2^{20} + w_{3,j+6} \cdot 2^{40}$
- $w_{3,j+8} - w_{1,j+7} = 0$
- $w_{5,j+8} - w_{0,j+7} = 0$
- $w_{0,j+10} \cdot w_{3,j+10} + p'_0 \cdot w_{0,j+8} - w_{6,j+8} + 2^{75} \cdot (w_{1,j+10} \cdot w_{3,j+10} + w_{0,j+10} \cdot w_{0,j+9} + p'_0 \cdot w_{1,j+8} + p'_1 \cdot w_{0,j+8}) - w_{5,j+8} \cdot 2^{75} - w_{7,j+9} \cdot 2^{135} = 0$
- $w_{2,j+10} \cdot w_{3,j+10} + w_{0,j+10} \cdot w_{1,j+9} + w_{1,j+10} \cdot w_{0,j+9} + p'_0 \cdot w_{2,j+8} + p'_2 \cdot w_{0,j+8} + p'_1 \cdot w_{1,j+8} - w_{4,j+8} + 2^{60} \cdot ((w_{4,j+10} \cdot 2^{40} + w_{5,j+10} \cdot 2^{20} + w_{6,j+10}) \cdot w_{3,j+10} + w_{0,j+10} \cdot (w_{3,j+9} \cdot 2^{40} + w_{4,j+9} \cdot 2^{20} + w_{5,j+9}) + w_{1,j+10} \cdot w_{1,j+9} + w_{2,j+10} \cdot w_{0,j+9} + p'_0 \cdot (w_{7,j+8} \cdot 2^{40} + w_{8,j+8} \cdot 2^{20} + w_{2,j+9}) + p'_3 \cdot w_{0,j+8} + p'_1 \cdot w_{2,j+8} + p'_2 \cdot w_{1,j+8}) - 2^{60} \cdot w_{3,j+8} + 2^{120} \cdot ((w_{4,j+10} \cdot 2^{40} + w_{5,j+10} \cdot 2^{20} + w_{6,j+10}) \cdot w_{0,j+9} + w_{1,j+10} \cdot (w_{3,j+9} \cdot 2^{40} + w_{4,j+9} \cdot 2^{20} + w_{5,j+9}) + w_{2,j+10} \cdot w_{1,j+9} + p'_1 \cdot (w_{7,j+8} \cdot 2^{40} + w_{8,j+8} \cdot 2^{20} + w_{2,j+9}) + p'_3 \cdot w_{1,j+8} + p'_2 \cdot w_{2,j+8}) + w_{7,j+9} - 2^{122} \cdot w_{8,j+9} = 0$
- $w_{4,j+11} = w_{5,j+11} \cdot 2^{58} + w_{6,j+11} \cdot 2^{38} + w_{7,j+11} \cdot 2^{18} + w_{8,j+10}$
- $w_{0,j+11} = w_{1,j+11} \cdot 2^{48} + w_{2,j+11} \cdot 2^{28} + w_{3,j+11} \cdot 2^8 + w_{7,j+10}$

- $(w_{0,j+10} + w_{1,j+10} \cdot 2^{75} + w_{2,j+10} \cdot 2^{135} + (w_{4,j+10} \cdot 2^{40} + w_{5,j+10} \cdot 2^{20} + w_{6,j+10}) \cdot 2^{195}) \cdot (w_{3,j+10} + w_{0,j+9} \cdot 2^{75} + w_{1,j+9} \cdot 2^{135} + (w_{3,j+9} \cdot 2^{40} + w_{4,j+9} \cdot 2^{20} + w_{5,j+9}) \cdot 2^{195}) = p \cdot (w_{0,j+8} + w_{1,j+8} \cdot 2^{75} + w_{2,j+8} \cdot 2^{135} + (w_{7,j+8} \cdot 2^{40} + w_{8,j+8} \cdot 2^{20} + w_{2,j+9}) \cdot 2^{195}) + w_{6,j+8} + w_{5,j+8} \cdot 2^{75} + w_{4,j+8} \cdot 2^{135} + w_{3,j+8} \cdot 2^{195}$

Lookup constraints:

- $(w_{1,j+0}), (w_{2,j+0}), (w_{3,j+0}), (w_{4,j+0}), (w_{5,j+0}), (w_{6,j+0}), (w_{7,j+0}), (w_{2,j+1}), (w_{3,j+1}), (w_{4,j+1}), (w_{5,j+1}), (w_{6,j+1}), (w_{7,j+1})$
- $(w_{1,j+2}), (w_{2,j+2}), (w_{3,j+2}), (w_{4,j+2}), (w_{5,j+2}), (w_{6,j+2}), (w_{7,j+2}), (w_{2,j+3}), (w_{3,j+3}), (w_{4,j+3}), (w_{5,j+3}), (w_{6,j+3}), (w_{7,j+3})$
- $(w_{1,j+4}), (w_{2,j+4}), (w_{3,j+4}), (w_{4,j+4}), (w_{5,j+4}), (w_{6,j+4}), (w_{7,j+4}), (w_{2,j+5}), (w_{3,j+5}), (w_{4,j+5}), (w_{5,j+5}), (w_{6,j+5}), (w_{7,j+5})$
- $(w_{1,j+6}), (w_{2,j+6}), (w_{3,j+6}), (w_{4,j+6}), (w_{5,j+6}), (w_{6,j+6}), (w_{7,j+6}), (w_{2,j+7}), (w_{3,j+7}), (w_{4,j+7}), (w_{5,j+7}), (w_{6,j+7}), (w_{7,j+7})$
- $(w_{1,j+11}), (w_{2,j+11}), (w_{3,j+11}), (w_{7,j+10}), (w_{7,j+10} \cdot 2^{12})$
- $(w_{5,j+11}), (w_{6,j+11}), (w_{7,j+11}), (w_{8,j+10}), (w_{8,j+10} \cdot 2^2)$

Copy constraints:

$$\begin{aligned} & (w_{0,j+8}, w_{0,j+4}), (w_{1,j+8}, w_{0,j+5}), (w_{2,j+8}, w_{1,j+5}), \\ & (w_{6,j+8}, w_{0,j+6}), (w_{7,j+8}, w_{1,j+4}), (w_{8,j+8}, w_{2,j+4}), (w_{0,j+9}, w_{0,j+3}), \\ & (w_{1,j+9}, w_{1,j+3}), (w_{2,j+9}, w_{3,j+4}), (w_{3,j+9}, w_{1,j+2}), (w_{4,j+9}, w_{2,j+2}), \\ & (w_{5,j+9}, w_{3,j+2}), (w_{7,j+9}, w_{0,j+11}), (w_{8,j+9}, w_{4,j+11}), (w_{0,j+10}, w_{0,j+0}), \\ & (w_{1,j+10}, w_{0,j+1}), (w_{2,j+10}, w_{1,j+1}), (w_{3,j+10}, w_{0,j+2}), (w_{4,j+10}, w_{1,j+0}), \\ & (w_{5,j+10}, w_{2,j+0}), (w_{6,j+10}, w_{3,j+0}) \end{aligned}$$

The proof of the addition of the numbers from  $\mathbb{F}_p$  proceeds as in the multiplication. We check an equation modulo  $k$  and  $2^t$ :

$$a + b = p \cdot q + r$$

We use the range proofs as above for  $a, b$ , and  $r$ . Since the value  $q$  can be equal to 0 or 1, we use the short-range check without any lookups. The second part of the proof can be implemented as the following:

1.  $(a \cdot b) \bmod k = (p \cdot q + r) \bmod k$
2.  $a_0 \cdot b_0 + p' \cdot q_0 - r_0 = v \cdot 2^3$ , where  $p'$  is  $-p \bmod 2^3$ .
3. Range-check that  $v \leq 2^{27}$ .

It is possible to extend to  $n < p$  additions. Thus, the value  $q$  is equal to an amount of additions minus 1,  $t = q + 2$ . The number of  $t_i$  is increased by depending on  $t$ . Particularly, the scalar multiplication proceeds as an extension of additions.

However, we need more special cases of non-native arithmetics for the elliptic curve's multiplication circuits.

1. Let  $a^2 \mp b^2 \mp c = p \cdot q + r$ , where  $c$  is constant. We change a range check for  $q$  to  $q < 2p$ . The total amount of the limbs does not change, but the last limb has to be checked by multiplication to  $2^4$ .
2. Let  $2 \cdot a \cdot b$ . This case is similar to the case from step 1.

### Complete addition circuit :

1.  $t_0 = x_1 \cdot y_2$  (7 rows)
2.  $t_1 = x_2 \cdot y_1$  (7 rows)
3.  $t_2 = x_1 \cdot x_2$  (7 rows)
4.  $t_3 = y_2 \cdot y_1$  (7 rows)
5.  $z_0 = t_0 + t_1$  (4 rows)
6.  $z_1 = t_2 + t_3$  (4 rows)
7.  $z_2 = t_0 \cdot t_1$  (7 rows)

8.  $k_0 = d \cdot z_2$  (7 rows)
9.  $k_1 = x_3 \cdot k_0$  (7 rows)
10.  $k_2 = y_3 \cdot k_0$  (7 rows)
11.  $k_3 = x_3 + k_1$  (4 rows)
12.  $k_4 = y_3 - k_2$  (4 rows)

**Fixed-base scalar multiplication circuit** : We precompute all values  $l \cdot (k \cdot B)$ , where  $k \in \{0, 2^{22}, 2^{44}, 2^{66}, 2^{88}, 2^{110}, 2^{132}, 2^{154}, 2^{176}, 2^{198}, 2^{220}, 2^{242}, 2^{253}\}$  and  $0 \leq l < 2^{22}$ . Since the size in bits of scalar is 253 we use 13 lookups. The first 12 lookups are used for  $l \cdot (k \cdot B)$  and the last one for range constraining. Thus, the 11 complete addition's constraints are required for fixed-base multiplication.

**Decomposition circuit** The decomposition circuit is a specific function for SHA-512, which prepares output to the non-native variable base scalar multiplication. Let  $\{k_0, k_1, k_2, \dots, k_7\}$  be a SHA-512 output. Suppose that we want to constrain  $k_0 + k_1 \cdot 2^{64} + \dots + k_7 \cdot 2^{448} = L \cdot q + r$ , where  $L = 2^{252} + 2774231777737235353851937790883648493$ . The size of each  $k_i$  is range-constrained by SHA-512 circuit. Since each degree of two can be reduced modulo  $L$  on the circuit definition's step, the value  $q$  is range-constrained by  $2^{67}$  and  $t = 69$ . Thus, the  $q$  decomposed to  $q_0, q_1, q_2, q_3$ , which corresponds to 20, 20, 20, 7 bits.

**Non-native decomposition circuit**

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$r_0$	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$r_7$	$r_8$
$j + 1$	$r_9$	$r_{10}$	$r_{11}$	$r_{12}$	$r$	$\xi$	$c$		
$j + 2$	$q_0$	$q_1$	$q_2$	$q_3$	$v_0$	$v_1$	$v_2$	$v_3$	
$j + 3$	$k_0$	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$v$

Let  $s_r$  be  $(w_{0,j+0} + w_{1,j+0} + w_{2,j+0} + w_{3,j+0} + w_{4,j+0} + w_{5,j+0} + w_{6,j+0} + w_{7,j+0} + w_{8,j+0}, w_{0,j+1}, w_{1,j+1} + w_{2,j+1} - 12 \cdot (2^{20} - 1))$ .

Constraints:

- $w_{0,j+3} + w_{1,j+3} \cdot 2^{64} + w_{2,j+3} \cdot 2^{128} + w_{3,j+3} \cdot 2^{192} + w_{4,j+3} \cdot (2^{256} \bmod k) + w_{5,j+3} \cdot (2^{320} \bmod k) + w_{6,j+3} \cdot (2^{384} \bmod k) + w_{7,j+3} \cdot (2^{448} \bmod k) - (w_{0,j+2} \cdot 2^{47} + w_{1,j+2} \cdot 2^{27} + w_{2,j+2} \cdot 2^7 + w_{3,j+2}) \cdot L + (w_{4,j+1}) = 0$
- $w_{4,j+1} = w_{3,j+1} + w_{2,j+1} \cdot 2^{13} + w_{1,j+1} \cdot 2^{33} + w_{0,j+1} \cdot 2^{53} + w_{8,j+0} \cdot 2^{73} + w_{7,j+0} \cdot 2^{93} + w_{6,j+0} \cdot 2^{113} + w_{5,j+0} \cdot 2^{133} + w_{4,j+0} \cdot 2^{153} + w_{3,j+0} \cdot 2^{173} + w_{2,j+0} \cdot 2^{193} + w_{1,j+0} \cdot 2^{213} + w_{0,j+0} \cdot 2^{233}$
- $s_r \cdot (w_{5,j+1} \cdot s_r - 1) = 0$
- $w_{5,j+1} \cdot (s_r) + (1 - w_{5,j+1} \cdot s_r) \cdot w_{6,j+1} - 1 = 0$
- $w_{0,j+3} + w_{1,j+3} \cdot 2^{64} + (w_{0,j+2} \cdot 2^{47} + w_{1,j+2} \cdot 2^{27} + w_{2,j+2} \cdot 2^7 + w_{3,j+2}) \cdot (-p \bmod 2^t) - (w_{3,j+1} + w_{2,j+1} \cdot 2^{13} + w_{1,j+1} \cdot 2^{33} + w_{0,j+1} \cdot 2^{53}) = v \cdot 2^{69}$
- $w_{8,j+3} = w_{4,j+2} \cdot 2^{41} + w_{5,j+2} \cdot 2^{21} + w_{6,j+2} \cdot 2 + w_{7,j+2}$
- $(w_{8,j+2} - 1) \cdot w_{8,j+2} = 0$

Lookup constraints:

- $(w_{0,j+0}), (w_{1,j+0}), (w_{2,j+0}), (w_{3,j+0}), (w_{4,j+0}), (w_{5,j+0}), (w_{6,j+0}), (w_{7,j+0}), (w_{8,j+0}), (w_{0,j+1}), (w_{1,j+1}), (w_{2,j+1}), (w_{3,j+1}), (w_{4,j+1}), (w_{5,j+1}), (w_{6,j+1}), (w_{7,j+1}), (w_{8,j+1})$
- $(w_{0,j+2}), (w_{1,j+2}), (w_{2,j+2}), (w_{3,j+2}), (w_{3,j+2} \cdot 2^{13})$
- $(w_{4,j+2}), (w_{5,j+2}), (w_{6,j+2}), (w_{7,j+2})$

**Variable-base scalar multiplication** :

The values  $b_i, i = 0, \dots, 252$  are binary representation of the scalar  $k'$ .

The values  $(x_1, y_1) = A$ .

$$(x_2, y_2) = 2(b_{252} \cdot (x_1, y_1)) + b_{251} \cdot (x_1, y_1)$$

$$(x_i, y_i) = 2(x_{i-1}, y_{i-1}) + b_{253-i} \cdot (x_1, y_1), \text{ for } i \in \{3, \dots, 253\}$$

For  $(x_i, y_i)$  the following is checked:

$$1. \ x_3 \cdot ((y_1^2 - x_1^2) \cdot (2 - y_1^2 + x_1^2) + 2dx_1y_1(y_1^2 + x_1^2) \cdot x_2y_2b) - (2x_1y_1 \cdot (2 - y_1^2 + x_1^2) \cdot (y_2b + (1 - b)) + (y_1^2 + x_1^2) \cdot (y_1^2 - x_1^2) \cdot x_2b)$$

$$2. \ y_3 \cdot ((y_1^2 - x_1^2) \cdot (2 - y_1^2 + x_1^2) - 2dx_1y_1(y_1^2 + x_1^2) \cdot x_2y_2b) - (2x_1y_1 \cdot (2 - y_1^2 + x_1^2) \cdot x_2b + (y_1^2 + x_1^2) \cdot (y_1^2 - x_1^2) \cdot (y_2b + (1 - b)))$$

This can be implemented in the following algorithm:

1.  $t_0 = (y_1^2 - x_1^2)$ . (11 rows)
2.  $t_1 = (2 - y_1^2 + x_1^2)$ . (11 rows)
3.  $t_2 = (t_0 \cdot t_1)$ . (11 rows)
4.  $t_3 = (y_1^2 + x_1^2)$ . (11 rows)
5.  $t_4 = 2 \cdot x_1 \cdot y_1$ . (11 rows)
6.  $t_5 = b \cdot x_2 \cdot y_2$ . (11 rows)
7.  $t_6 = t_3 \cdot t_4$ . (11 rows)
8.  $t_7 = t_6 \cdot t_3$ . (11 rows)
9.  $t_8 = d \cdot t_7$ . (9 rows)
10.  $t_9 = (t_8 + t_2) \cdot x_3$ . (13 rows)
11.  $z_0 = t_4 \cdot t_1$ . (11 rows)
12.  $z_1 = z_0 \cdot (y_2 \cdot b + (1 - b))$ . (11 rows)
13.  $z_2 = t_3 \cdot t_0$ . (11 rows)
14.  $z_3 = b \cdot z_2 \cdot x_2$ . (11 rows)
15.  $t_9 - z_3 == z_1$ . (7 rows)
16.  $c_0 = y_3 \cdot (t_2 - t_8)$ . (13 rows)
17.  $d_0 = b \cdot z_0 \cdot x_2$ . (11 rows)
18.  $d_1 = z_2 \cdot (y_2 \cdot b + (1 - b))$ . (11 rows)
19.  $c_0 - d_0 == d_1$ . (7 rows)

As another option, we add a protocol description with only non-native multiplications, additions, and subtractions without any special cases:

1.  $s_0 = x_1^2$ .
2.  $s_1 = y_1^2$
3.  $s_2 = x_1 \cdot y_1$
4.  $s_3 = b \cdot y_2$
5.  $s_4 = b \cdot x_2$
6.  $t_0 = s_0 + s_1$
7.  $t_1 = s_1 - s_0$
8.  $t_2 = s_0 - s_1$
9.  $t_3 = s_3 \cdot x_2$
10.  $t_4 = 2 \cdot s_2$
11.  $t_5 = d \cdot t_4$
12.  $l_0 = 2 + t_2$
13.  $l_1 = l_0 \cdot t_1$

14.  $l_2 = t_5 \cdot t_0$
15.  $l_3 = l_2 \cdot t_3$
16.  $r_0 = l_0 \cdot t_4$
17.  $r_1 = s_3 + (1 - b)$
18.  $r_2 = r_1 \cdot r_0$
19.  $r_3 = t_0 \cdot t_1$
20.  $r_4 = r_3 \cdot s_4$
21.  $p_0 = l_1 + l_3$
22.  $p_1 = l_1 - l_3$
23.  $p_2 = x_3 \cdot p_0$
24.  $p_3 = y_3 \cdot p_0$
25.  $z_0 = r_4 + r_2$
26.  $z_1 = r_0 \cdot s_4$
27.  $z_2 = r_3 \cdot r_1$
28.  $z_4 = z_1 + z_2$

Thus, it costs 203 rows per bit. Totally, it is  $50953 + 4$  rows.

### 2.6.8 Redshift Verification

#### WIP

Redshift circuit repeats all steps from Section 2.5.2. 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 from Section 2.6.4 is used to check hash operations correctness.

To check polynomial interpolation, the following circuit is used:

	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$j + 0$	$a_0$	$a_1$	$s_0$	$s_1$	$x$	$y$	$\alpha$	$\beta$	$\dots$

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

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

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  $f(X) = \alpha \cdot X + \beta$ .

### 2.6.9 Validator Set Proof Circuit

Let  $E_1$ ,  $E_2$  are two consecutive epochs.  $E_1$  is confirmed on the Ethereum side and  $E_2$  is not. To prove the validator set  $S_2$  of  $E_2$ , the prover does the following:

1. for  $i$  from  $v_1, \dots, v_m = S$ :
  - 1.1 Show the inclusion proof of the last stake-change transaction to  $v_i$  (no later than the end of  $E_1$ ).
  - 1.2 Show that there wasn't any stake update since the last delegate transaction  $tx_{last}$ . That means verifying all transactions between  $tx_{last}$  and the beginning of  $E_2$ .

Such an approach provides additional overhead to the prover. The proof of correct validator set will be simplified after implementation of the "Simple Payment and State Verification"<sup>4</sup> proposal.

This leads to the circuit defined as follows:

WIP

---

<sup>4</sup><https://docs.solana.com/proposals/simple-payment-and-state-verification>



# 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

Verification contains the following steps:

1. Get input: proof  $\pi$  and new state  $S$ .
2. Verify placeholder proof  $\pi$  (see placeholder verification below).
3. Update the last confirmed Solana state with  $S$  (see 3.1.1).

**Placeholder verification** part contains the following components:

1. **Proof Deserialization:** Handles the input data processing (marshalling/demarshalling) mechanisms.

These mechanisms are defined within the `*_marshalling`-postfixed files.

- [https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/basic\\_marshalling.sol](https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/basic_marshalling.sol)
- [https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/basic\\_marshalling\\_calldata.sol](https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/basic_marshalling_calldata.sol)

2. **Parameters Initialization** Processes commitment scheme (FRI) parameters from the input data <https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/types.sol>

3. **Proof Verification:** Includes a verification of the hash-based commitment scheme and the proof itself.

The verification itself is defined within the directory `components`, each of which<sup>1</sup> defines a set of gates relevant to particular component. Verification algorithm contains:

- Transcript (Fiat-Shamir transformation to non-interactive protocol) <https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/cryptography/transcript.sol>
- Permutation argument: [https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/placeholder/permutation\\_argument.sol](https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/placeholder/permutation_argument.sol)

---

<sup>1</sup>For instance, [https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/placeholder/verifier\\_non\\_native\\_field\\_add\\_component.sol](https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/placeholder/verifier_non_native_field_add_component.sol)

- Gate Argument depends on the circuit definition and is unique for each circuit.  
[https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/components/non\\_native\\_field\\_add\\_gen.sol](https://github.com/NilFoundation/evm-placeholder-verification/blob/master/contracts/components/non_native_field_add_gen.sol)
- Commitment Scheme verification  
<https://github.com/NilFoundation/evm-placeholder-verification/tree/master/contracts/commitments>

### 3.1.1 State Proof Sequence Maintenance

To verify the validator set within the state proof submitted is derived from original Solana's genesis data, it is supposed to maintain validator's set state proofs sequence on in-EVM side in a data structure as follows.

Let  $B_{n_1}$  be the last state confirmed on Ethereum. Let us say some prover wants to confirm a new  $B_{n_2}$  state. Denote by  $H_B$  the hash of a state  $B$ . So a Merkle Tree  $T_{n_1, n_2}$  from the set  $\{H_{B_{n_1}}, \dots, H_{B_{n_2}}\}$

The state proof sequence correctness statement contains (but not bounded by) the following points:

---

#### Algorithm 1 Proving Statement

---

1. Show that the validator set is correct.
  2. Show that the  $B_{n_1}$  corresponds to the last confirmed state on Ethereum.
  3. for  $i$  from the interval  $[n_1 + 1, n_2 - 1]$ :
    - 3.1 Show that  $B_i$  contains  $H_{B_{i-1}}$  as a hash of the previous state.
  4. for  $i$  from the interval  $[n_2, n_2 + 32]$ :
    - 4.1 Show that  $B_i$  contains  $H_{B_{i-1}}$  as a hash of the previous state.
    - 4.2 Show that there are enough valid signatures from the current validator set for  $B_i$ .
  5. Build a Merkle Tree  $T_{n_1, n_2}$  from the set  $\{H_{B_{n_1}}, \dots, H_{B_{n_2}}\}$ .
- 

$T_{n_1, n_2}$  allows to provide a successful transaction from  $\{B_{n_1}, \dots, B_{n_2}\}$  to the Ethereum-based proof verifier later.

## 3.2 Verification Logic API Reference

Every call to Placeholder public API verification function eventually leads to a call of internal verification function for chosen circuit (for example, [https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/verifier\\_unified\\_addition\\_component.sol#L66](https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/verifier_unified_addition_component.sol#L66)). These internal verification functions should be supplied with proof bytblob and initialized verification parameters.

For now there is test public API which execute basic logic consisting of:

1. parsing proof bytblob
2. verification parameters initialization
3. circuit specific internal verification function calling
4. verification result returning

Example of test public API function declaration intended for verification of unified addition circuit (see [https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public\\_api\\_placeholder\\_unified\\_addition\\_component.sol#L34](https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public_api_placeholder_unified_addition_component.sol#L34)):

```
function verify(
    bytes calldata blob,
    uint256[] calldata init_params,
```

```

    int256[][] calldata columns_rotations
) public {...}

```

More details regarding public API input data structure see in the next section.  
Other existing test public API functions could be found here:

- [https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public\\_api\\_placeholder\\_non\\_native\\_field\\_add\\_component.sol#L34](https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public_api_placeholder_non_native_field_add_component.sol#L34)
- [https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public\\_api\\_placeholder\\_variable\\_base\\_scalar\\_mul\\_component.sol#L34](https://github.com/NilFoundation/evm-placeholder-verification/blob/7672050e4bcd9e98f90ddda8a85f7c81d0ecc14/contracts/placeholder/test/public_api_placeholder_variable_base_scalar_mul_component.sol#L34)

### 3.3 Input Data Structures

All input data divided into two parts:

1. Placeholder proof byteblob itself;
2. Verification parameters used to verify proof.

#### 3.3.1 Placeholder Proof Structure

Placeholder proof consists of different fields and some of them are of complex structure types, which will be described in top-down order.

So, the first one Placeholder proof has the following structure, which is described in pseudocode:

```

struct PlaceholderProof {
    witness_commitment: vector<uint8>
    v_perm_commitment: vector<uint8>
    input_perm_commitment: vector<uint8>
    value_perm_commitment: vector<uint8>
    v_l_perm_commitment: vector<uint8>
    T_commitment: vector<uint8>
    challenge: uint256
    lagrange_0: uint256
    witness: LPCProof
    permutation: LPCProof
    quotient: LPCProof
    lookups: vector<LPCProof>
    id_permutation: LPCProof
    sigma_permutation: LPCProof
    public_input: LPCProof
    constant: LPCProof
    selector: LPCProof
    special_selectors: LPCProof
}

```

In turn proof of LPC algorithm has the following structure:

```

struct LPCProof {
    T_root: vector<uint8>
    z: vector<vector<uint256>>
    fri_proofs: vector<FRIPProof>
}

```

The next one description is for structure of FRI algorithm proof:

```

struct FRIPProof {
    final_polynomials: vector<vector<uint256>>
    round_proofs: vector<FRIRoundProof>
}

```

One of the components of the FRI algorithm proof is so called round FRI proof, which has the following structure:

```
struct FRIRoundProof {
    colinear_value: vector<uint256>
    T_root: vector<uint256>
    colinear_path: MerkleProof
    p: vector<MerkleProof>
}
```

The next important component is the merkle tree proof of the following structure:

```
struct MerkleProof {
    leaf_index: uint64
    root: vector<uint8>
    path: vector<MerkleProofLayer>
}

struct MerkleProofLayer {
    layer: vector<MerkleProofLayerElement>
}
```

In the simplest and used case of the merkle tree with arity 2 layer consists of only one element:

```
struct MerkleProofLayerElement {
    position: uint64
    hash: vector<uint8>
}
```

It is important to note that before sending Placeholder proof to EVM for verification it should be serialized into byteblob format, which is done using corresponding marshalling module (<https://github.com/NilFoundation/crypto3-zk-marshalling/blob/01b531550a99232586e17c1e383e4693a4ddc924/include/nil/crypto3/marshalling/zk/types/placeholder/proof.hpp>).

### 3.3.2 Verification Parameters

Verification parameters are used to parametrize Placeholder algorithm depending on chosen security parameters and specific circuit for which proof was created.

Following parameters are required to complete Placeholder verification procedure in-EVM:

```
modulus: uint256 - modulus of chosen prime field
r: uint256 - parameter of FRI algorithm
max_degree: uint256 - parameter of FRI algorithm
lambda: uint256 - parameter of LPC algorithm
rows_amount: uint256 - parameter defined by chosen circuit
omega: uint256 - parameter defined by chosen circuit
max_leaf_size: uint256 - parameter dependent on specific instance of Placeholder algorithm, etc.
Domains generators: vector<uint256> - parameter defined by chosen circuit
q polynomial: vector<uint256> - parameter of Placeholder algorithm
Columns rotations: vector<vector<int256>> - parameter defined by chosen circuit
```

# 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](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>.