In-EVM Solana State Verification
Circuit Description

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

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

October 11, 2021

1 Introduction

This paper contains a description of the following PLONK-style circuits:

- SHA256 for block headers and transactions hashes verification.
- EdDSA for validators signature verification.
- Poseidon/Reinforce Concrete for state Merkle-tree and proof generation.

2 SHA256 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 - 1					0
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:

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

The range proofs are included for each input data block.

The function σ_0 contain sparse mapping subcircuit with base 2. Let a be divided to a_0, a_1, a_2, a_3 8 bits-chunks. 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 NORMALIZE2**: Read a'_i to a_i
2. **SHA-256 8ROT7 32**: Read a'_0 to r_1
3. **SHA-256 8ROT2 32**: Read a'_2 to r_2

4. **SHA-256 8ROT3 32**: Read a'_0 to r_3

	w_1	w_2	w_3	w_4	w_o
j - 8					$a_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2					r_1
j + 3					r_2
j + 4					r_3
j + 5	a_{rot7}	a_{rot18}	a_{rot3}		σ_0

Sparse map gate constraints:

$$
\begin{gathered}
w_{o,j} = w_{1,j} + w_{2,j} * 2^8 + w_{3,j} * 2^{8*2} + w_{4,j} * 2^{8*3} \\
w_{o,j+1} = w_{1,j+1} + w_{2,j+1} * 4^8 + w_{3,j+1} * 4^{8*2} + w_{4,j+1} * 4^{8*3} \\
w_{1,j+5} = w_{o,j+2} + w_{2,j+1} * 4^{8-7} + w_{3,j+1} * 4^{8*2-7} + w_{4,j+1} * 4^{8*3-7} \\
w_{2,j+5} = w_{o,j+3} + w_{1,j+1} * 4^{8*2-2} + w_{2,j+1} * 4^{8*3-2} + w_{4,j+1} * 4^{8-2} \\
w_{3,j+5} = w_{o,j+4} + w_{2,j+1} * 4^{8-3} + w_{3,j+1} * 4^{8*2-3} + w_{4,j+1} * 4^{8^3-3} \\
w_{o,j+5} = w_{1,j+5} + w_{2,j+5} + w_{3,j+5}
\end{gathered}
$$

The function σ_1 contain sparse mapping subcircuit with base 2. Let a be divided to a_0, a_1, a_2, a_3 8 bits-chunks. 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 NORMALIZE2**: 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 - 8					$a_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2					r_1
j + 3					r_2
j + 4					r_3
j + 5	a_{rot17}	a_{rot19}	a_{rot10}		σ_1

Sparse map gate constraints:

$$
\begin{gathered}
w_{o,j} = w_{1,j} + w_{2,j} * 2^8 + w_{3,j} * 2^{8*2} + w_{4,j} * 2^{8*3} \\
w_{o,j+1} = w_{1,j+1} + w_{2,j+1} * 4^8 + w_{3,j+1} * 4^{8*2} + w_{4,j+1} * 4^{8*3} \\
w_{1,j+5} = w_{o,j+2} + w_{1,j+1} * 4^{8*2-1} + w_{2,j+1} * 4^{8*3-1} + w_{4,j+1} * 4^{8-1} \\
w_{2,j+5} = w_{o,j+3} + w_{1,j+1} * 4^{8*2-3} + w_{2,j+1} * 4^{8*3-3} + w_{4,j+1} * 4^{8-3} \\
w_{3,j+5} = w_{o,j+4} + w_{1,j+1} * 4^{8^3-2} + w_{3,j+1} * 4^{8-2} + w_{4,j+1} * 4^{8^2-2} \\
w_{o,j+5} = w_{1,j+5} + w_{2,j+5} + w_{3,j+5}
\end{gathered}
$$

The sparse values σ_0 and σ_1 have to be normalized. We use **SHA256 NORMALIZE2**

	w_1	w_2	w_3	w_4	w_o
j - 8					$a_{0,2}$
...					
j + 0	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	acc
j + 1	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	σ_i

Normalize gate constraints:

$$
w_{o,i} = w_{o,i-1} * 8^4 + w_{4,i} * 8^3 + w_{3,i} * 8^2 + w_{2,i} * 8 + w_{1,i}
$$

The final addition requires one add gate.

The Σ_0 function contain sparse mapping subcircuit with base 2. Let a be divided to a_0, a_1, a_2, a_3 8 bits-chunks. 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 NORMALIZE2**: 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 - 8					$a_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2					r_1
j + 3					r_2
j + 4					r_3
j + 5	a_{rot2}	a_{rot13}	a_{rot22}		σ_0

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + w_{2,j} * 2^8 + w_{3,j} * 2^{8*2} + w_{4,j} * 2^{8*3}$$
$$w_{o,j+1} = w_{1,j+1} + w_{2,j+1} * 4^8 + w_{3,j+1} * 4^{8*2} + w_{4,j+1} * 4^{8*3}$$
$$w_{1,j+5} = w_{o,j+2} + w_{2,j+1} * 4^{8-2} + w_{3,j+1} * 4^{8*2-2} + w_{4,j+1} * 4^{8*3-2}$$
$$w_{2,j+5} = w_{o,j+3} + w_{1,j+1} * 4^{8*3-5} + w_{3,j+1} * 4^{8-5} + w_{4,j+1} * 4^{8*2-5}$$
$$w_{3,j+5} = w_{o,j+4} + w_{1,j+1} * 4^{8*2-6} + w_{2,j+1} * 4^{8*3-6} + w_{4,j+1} * 4^{8-6}$$
$$w_{o,j+5} = w_{1,j+5} + w_{2,j+5} + w_{3,j+5}$$

The Σ_1 function contain sparse mapping subcircuit with base 2. Let a be divided to a_0, a_1, a_2, a_3 8 bits-chunks. 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 NORMALIZE2**: 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 - 8					$a_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2					r_1
j + 3					r_2
j + 4					r_3
j + 5	a_{rot2}	a_{rot13}	a_{rot22}		σ_0

Sparse map gate constraints:

$$w_{o,j} = w_{1,j} + w_{2,j} * 2^8 + w_{3,j} * 2^{8*2} + w_{4,j} * 2^{8*3}$$
$$w_{o,j+1} = w_{1,j+1} + w_{2,j+1} * 4^8 + w_{3,j+1} * 4^{8*2} + w_{4,j+1} * 4^{8*3}$$
$$w_{1,j+5} = w_{o,j+2} + w_{2,j+1} * 4^{8-6} + w_{3,j+1} * 4^{8*2-6} + w_{4,j+1} * 4^{8*3-6}$$
$$w_{2,j+5} = w_{o,j+3} + w_{1,j+1} * 4^{8*3-3} + w_{3,j+1} * 4^{8-3} + w_{4,j+1} * 4^{8*2-3}$$
$$w_{3,j+5} = w_{o,j+4} + w_{1,j+1} * 4^{8-1} + w_{2,j+1} * 4^{8*2-1} + w_{3,j+1} * 4^{8*3-1}$$
$$w_{o,j+5} = w_{1,j+5} + w_{2,j+5} + w_{3,j+5}$$

The sparse values Σ_0 and Σ_1 have to be normalized. We use **SHA256 NORMALIZE2**

	w_1	w_2	w_3	w_4	w_o
j - 8					$a_{0,2}$
...					
j + 0	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	acc
j + 1	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	Σ_i

Normalize gate constraints:

$$w_{o,i} = w_{o,i-1} * 8^4 + w_{4,i} * 8^3 + w_{3,i} * 8^2 + w_{2,i} * 8 + w_{1,i}$$

The Maj function contain sparse mapping subcircuit with base 2 for a, b, c. Let $a; b; c$ be divided to $a_0, a_1, a_2, a_3; b_0, b_1, b_2, b_3; c_0, c_1, c_2, c_3$ 8 bits-chunks. 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 NORMALIZE2**: Read a'_i to a_i

Similarly for b and c.

	w_1	w_2	w_3	w_4	w_o
j - 24					$a_{0,2}$
...					
j - 16					$b_{0,2}$
...					
j - 8					$c_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2	b_0	b_1	b_2	b_3	b
j + 3	b'_0	b'_1	b'_2	b'_3	b'
j + 4	c_0	c_1	c_2	c_3	c
j + 5	c'_0	c'_1	c'_2	c'_3	c'
j + 6					maj

Sparse map gate constraints:

$$w_{o,i} = w_{1,i} + w_{2,i} * 2^8 + w_{3,i} * 2^{8*2} + w_{4,i} * 2^{8*3}$$
$$w_{o,i+1} = w_{1,i+1} + w_{2,i+1} * 4^8 + w_{3,i+1} * 4^{8*2} + w_{4,i+1} * 4^{8*3}$$
$$w_{o,j+5} = w_{o,j+1} + w_{o,j+3} + w_{o,j+5}$$

The sparse values maj have to be normalized. We use **SHA256 MAJ NORMALIZE2**

	w_1	w_2	w_3	w_4	w_o
j - 8					$a_{0,2}$
...					
j + 0	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	acc
j + 1	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	maj

Normalize gate constraints:

$$w_{o,i} = w_{o,i-1} * 8^4 + w_{4,i} * 8^3 + w_{3,i} * 8^2 + w_{2,i} * 8 + w_{1,i}$$

The final addition requires one add gate.

The Ch function contain sparse mapping subcircuit with base 2 for a, b, c. Let $a; b; c$ be divided to $a_0, a_1, a_2, a_3; b_0, b_1, b_2, b_3; c_0, c_1, c_2, c_3$ 8 bits-chunks. 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

Similarly for b and c.

	w_1	w_2	w_3	w_4	w_o
j - 24					$a_{0,2}$
...					
j - 16					$b_{0,2}$
...					
j - 8					$c_{0,2}$
...					
j + 0	a_0	a_1	a_2	a_3	a
j + 1	a'_0	a'_1	a'_2	a'_3	a'
j + 2	b_0	b_1	b_2	b_3	b
j + 3	b'_0	b'_1	b'_2	b'_3	b'
j + 4	c_0	c_1	c_2	c_3	c
j + 5	c'_0	c'_1	c'_2	c'_3	c'
j + 6					ch

Sparse map gate constraints:

$$w_{o,i} = w_{1,i} + w_{2,i} * 2^8 + w_{3,i} * 2^{8*2} + w_{4,i} * 2^{8*3}$$
$$w_{o,i+1} = w_{1,i+1} + w_{2,i+1} * 7^8 + w_{3,i+1} * 7^{8*2} + w_{4,i+1} * 7^{8*3}$$
$$w_{o,j+5} = w_{o,j+1} + 2 * w_{o,j+3} + 3 * w_{o,j+5}$$

The sparse values ch have to be normalized. We use **SHA256 CH NORMALIZE7**

	w_1	w_2	w_3	w_4	w_o
j - 8					$a_{0,2}$
...					
j + 0	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	acc
j + 1	$a_{2,1}$	$a_{2,2}$	$a_{3,1}$	$a_{3,2}$	ch

Normalize gate constraints:

$$w_{o,i} = w_{o,i-1} * 8^4 + w_{4,i} * 8^3 + w_{3,i} * 8^2 + w_{2,i} * 8 + w_{1,i}$$

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.

2.1 SHA-512

WIP

3 Poseidon Circuit

WIP
For now, there are two SNARK-friendly hash function candidates: Poseidon[1] and Reinforce Concrete[2].

4 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 $T_i = H_i$ // just notation for simplicity, not a real part of the circuit
4. for i from 0 to $k - 1$:

[1] https://www.poseidon-hash.info
[2] https://www.rc-hash.info

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 3

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

5 Ed25519 Circuit

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

1. Prove that s in the defined range.
2. $k ==$ SHA-512$(data||R||A||M)$ //See section 2.1
3. $8sB? =?8R + 8kA$ //See section 5.1

5.1 The Arithmetic of Elliptic Curves

Variable-base scalar multiplication circuit per bit b:

1. $b^2 = b$
2. $(y_1)(2b-1) = (y_2)$
3. $(x_2 - x_3)(\lambda_1) = (y_2 - y_3)$
4. $(B\lambda_1)(\lambda_1) = (A + x_3 + x_2 + x_4)$
5. $(x_3 - x_4)(\lambda_1 + \lambda_2) = (2y_3)$
6. $(B\lambda_2)(\lambda_2) = (A + x_4 + x_3 + x_5)$
7. $(x_3 - x_5)(\lambda_2) = (y_5 + y_3)$

EC Point Addition circuit:

1. $(x_2 - x_1)(y_3 + y_1) - (y_1 - y_2)(x_1 - x_3)$
2. $(x_1 + x_2 + x_3)(x_1 - x_3)(x_1 - x_3) - (y_3 + y_1)(y_3 + y_1)$

6 The Correct Validator Set Proof Circuit

WIP

7 Bringing it all together

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 6
3. for i from $n_1 + 1$ to $n_2 + 32$:

   3.1 $H_{B_i} = \texttt{sha256}(\texttt{block\_data}||H_{B_{i-1}})$ // see Section 2

4. for j from 0 to M:

   4.1 Ed25519 constraints for $H_{B_{n_2+32}}$ // see Section 5

5. Merkle tree constraints for the set $\{H_{B_{n_1}}, ..., H_{B_{n_2}}\}$ // see Section 4

References