Poseidon

General Notation

Types

$x : \mathbb{T}$
A variable x having type $\mathbb{T}$.

$v : \mathbb{T}^{[n]}$
An array of n elements each of type $\mathbb{T}$.

$m : \mathbb{T}^{[n \times n]}$
An $n \times n$ matrix having elements of type $\mathbb{T}$.

$\mathcal{I}_n : \mathbb{T}^{[n \times n]}$
The $n \times n$ identity matrix.

$b : \{0, 1\}$
A bit b.

$\textsf{bits} : \{0, 1\}^{[n]}$
An array **bits** of n bits.

$x : \mathbb{Z}_p$
A prime field element x.

$x \in \mathbb{Z}_n$
An integer in $x \in [0, n)$.

$x : \mathbb{Z}_{\geq 0}$
A non-negative integer.

$x : \mathbb{Z}_{>0}$
A positive integer.

$[n]$
The range of integers $0, \ldots, n-1$.

$[a, b)$
The range of integers $a, \ldots, b-1$.

Array Operations

All arrays and matrices are indexed starting at zero. An array of n elements has indices $0, \ldots, n-1$.

$v[i]$
Returns the i^{th} element of array v. When the notation $v[i]$ is cumbersome v_i is used instead.

$v[i..j]$
Returns a slice of the array v: $[v[i], \ldots, v[j-1]]$.

$v \parallel w$
Concatenates two arrays v and w, of types $\mathbb{T}^{[m]}$ and $\mathbb{T}^{[n]}$ respectively, producing an array of type $\mathbb{T}^{[m+n]}$. The above is equivalent to writing $[v_0, \ldots, v_{m-1}, w_0, \ldots, w_{n-1}]$.

$[f(\ldots)]_{i \in \{1,2,3\}}$
Creates an array using list comprehension; each element of the array is the output of the expression $f(\ldots)$ at each element of the input sequence (e.g. $i \in \{1, 2, 3\}$).

$v \vec{\oplus} w$
The element-wise field addition of two equally lengthed vectors whose elements are field elements: $v \vec{\oplus} w = [v_0 \oplus w_0, \ldots, v_{n-1} \oplus w_{n-1}]$.

Matrix Operations

$m_{i,j}$
Returns the value of matrix m at row i column j.

$m_{i,*}$
Returns the i^{th} row of matrix m.

$m_{*,j}$
Returns the j^{th} column of matrix m.

$$m_{1..,1..} = \begin{bmatrix} m_{1,1} & \cdots & m_{1,c-1} \\ \vdots & \ddots & \vdots \\ m_{r-1,1} & \cdots & m_{r-1,c-1} \end{bmatrix}$$

Returns a submatrix of m which excludes m's first row and first column (here m is an $r \times c$ matrix).

m^{-1}
The inverse of a square $n \times n$ matrix m, i.e. $m \times m^{-1} = \mathcal{I}_n$.

$$v \times m = [v_0, \ldots, v_{r-1}] \begin{bmatrix} m_{0,0} & \cdots & m_{0,c-1} \\ \vdots & \ddots & \vdots \\ m_{r-1,0} & \cdots & m_{r-1,c-1} \end{bmatrix} = [v \cdot m_{*,i}]_{i \in [c]}$$

Vector-matrix multiplication where vector v is a row vector, m is a matrix, and $\mathbf{len}(v) = \mathbf{rows}(m)$. The product is a row vector whose length is equal to the number of columns of m. The i^{th} element of the product vector is the dot product of v and the i^{th} column of m. In the above example v has length r and m is an $r \times c$ matrix.

$$m \times v = \begin{bmatrix} m_{0,0} & \cdots & m_{0,c-1} \\ \vdots & \ddots & \vdots \\ m_{r-1,0} & \cdots & m_{r-1,c-1} \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{c-1} \end{bmatrix} = \begin{bmatrix} m_{0,*} \cdot v \\ \vdots \\ m_{r-1,*} \cdot v \end{bmatrix}$$

Matrix-vector multiplication where vector v is a column vector, m is a matrix, and $\mathbf{len}(v) = \mathbf{columns}(m)$. The product is a column vector whose length is equal to the number of rows of m. The i^{th} element of the product vector is the dot product of the i^{th} row of m with v. In the above example v has length c and m is an $r \times c$ matrix.

Note: $v \times m = (m \times v)^T$ when m is symmetric $m = m^T$, i.e. the row vector-matrix product contains the same elements as the column vector-matrix product when m is symmetric.

Field Arithmetic

$a \oplus b$
Addition in $\mathbb{Z}_p$ of two field elements a and b.

x^α
Exponentiation in $\mathbb{Z}_p$ of a field element x to a positive integer power $\alpha : \mathbb{Z}_{\geq 0}$.

Bitwise Operations

$\oplus_{\text{xor}}$
Bitwise XOR.

$\bigoplus_{\text{xor } x \in \{1,2,3\}} x$
XOR's all values of a sequence. The above is equivalent to writing $1 \oplus_{\text{xor}} 2 \oplus_{\text{xor}} 3$.

Bitstrings

$[1, 0, 0] = 100_2$
A bit array can be written as an array $[1, 0, 0]$ or as a bitstring 100_2. The leftmost bit of the bitstring corresponds to the first bit in the array.

Binary-Integer Conversions

x **as** $\{0,1\}^{[n]}_{\text{msb}}$
Converts an intger $x : \mathbb{Z}_{\geq 0}$ into its n-bit binary representation. The most-significant bit (**msb**) is first (leftmost) in the produced bit array $\{0,1\}^{[n]}$, e.g. 6 **as** $\{0,1\}^{[3]}_{\text{msb}} = [1,1,0] = 110_2$.

bits$_{\text{msb}}$ **as** $\mathbb{Z}_{\geq 0}$
Converts a bit array **bits** : $\{0,1\}^{[n]}$ into an integer. The first bit in **bits** is the most significant (**msb**).

Poseidon Symbols

p
The prime field modulus.

M
The security level (measured in bits). M is roughly half the field size $M \approx \log_2(p)/2$.

t
The *width*; the number of field elements in a Poseidon instance's internal state array **state**.

(p, M, t)
A Poseidon instance. Each instance is fully specified using this parameter triple.

α
The S-box function's exponent $S(x) = x^\alpha$, where $\gcd(\alpha, p-1) = 1$.

R_F
The number of full rounds.

R_P
The number of partial rounds.

$R = R_F + R_P$
The total number of rounds

$R_f = \lfloor R_F/2 \rfloor$
Half the number of full rounds.

$r \in [R]$
The index of a round.

$r \in [R_f]$
The round index for a first-half full round.

$r \in [R_f, R_f + R_P)$
The round index for a partial round.

$r \in [R_f + R_P, R)$
The round index for a second-half full round.

state
A Poseidon instance's internal state array of t field elements $\mathbb{Z}_p$ which are transformed in each round.

RoundConstants
The round constants for an unoptimized Poseidon instance.

RoundConstants_r
The round constants that are added to Poseidon's **state** array before the S-boxes in round r of an unoptimized Poseidon instance.

$\text{RoundConstants}'$
The round constants for an optimized Poseidon instance.

$\text{RoundConstants}'_{\text{pre}}$
The round constants that are added to Poseidon's **state** array before the S-boxes in the first round $r = 0$ of an optimized Poseidon instance.

$\text{RoundConstants}'_r$
The round constants that are added to Poseidon's **state** array after the S-boxes in round r in an optimized Poseidon instance.

$\mathcal{M}$
The MDS matrix for a Poseidon instance.

$\mathcal{P}$
The *pre-sparse* matrix used in MDS mixing for the last first-half full round ($r = R_f - 1$) of the optimized Poseidon algorithm.

$\mathcal{S}$
An array of matrices used in MDS mixing for the partial rounds $r \in [R_f, R_f + R_P)$ of the optimized Poseidon algorithm.

Instantiation

The parameter triple (p, M, t) fully specifies a unique instance of Poseidon (a hash function that uses the same constants and parameters and performs the same operations). All other Poseidon parameters and constants are derived from the instantiation parameters.

The S-box exponent α is derived from the field modulus p such that $a \in \{3, 5\}$ and $\gcd(\alpha, p - 1) = 1$.

The round numbers R_F and R_P are derived from the field size and security level $(\lceil \log_2(p) \rceil, M)$.

The RoundConstants are derived from (p, M, t).

The MDS matrix $\mathcal{M}$ is derived from the width t.

The allowed preimage sizes are $\mathbf{len}(\text{preimage}) \in [1, t)$.

The total number of operations executed per hash is determined by the width and number of rounds (t, R_F, R_P).

Poseidon in Filecoin

$$\begin{aligned} p &= 52435875175126190479447740508185965837690552500527637822603658699938581184513 \\ &= \text{0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9feffffffffa} \end{aligned}$$

The prime field modulus in base-10 and base-16. Filecoin uses BLS12-381's scalar field for the Poseidon prime field $\mathbb{Z}_p$. Filecoin's modulus p is equal to the order of the BLS12-381 curve group's prime order subgroup $\mathbb{G}1$.

$M = 128 \text{ Bits}$
Filecoin targets the 128-bit security level (M is roughly half the field size $M \approx \log_2(p)/2$ where $\log_2(p) \approx 256$).

$t = \text{preimage length} + \text{output length} = \text{preimage length} + 1 = \text{Arity} + 1$
$t \in \{3, 5, 9, 12\}$
The size in field elements of Poseidon's internal state. Filecoin's Poseidon instances take preimages of varying lengths (2, 4, 8, and 11 field elements) and always return one field element.

The Filecoin protocol uses different Poseidon widths for different applications:

- $t = 3$ is used to hash 2:1 Merkle trees (*BinTrees*) and to derive SDR-PoRep's CommR
- $t = 5$ is used to hash 4:1 Merkle trees (*QuadTrees*)
- $t = 9$ is used to hash 8:1 Merkle trees (*OctTrees*)
- $t = 12$ is used to hash SDR-PoRep *columns* of 11 field elements

$\alpha = 5$
The S-box function's $S(x) = x^\alpha$ exponent. It is required that α is relatively prime to $p - 1$, i.e. $\gcd(\alpha, p - 1) = 1$, which is true for Filecoin's choice of p.

Round Numbers

The Poseidon round numbers are the number of full and partial rounds (R_F, R_P) for a Poseidon instance (p, M, t). The round numbers are chosen such that they minimize the total number of S-boxes:

$$\text{Number of S-boxes} = tR_F + R_P$$

while providing security against known attacks (statistical, interpolation, and Gröbner basis).

R_F and R_P are calculated using either the Python script `calc_round_numbers.py` (https://extgit.iaik.tugraz.at/krypto/hadeshash/-/blob/9d80ec0473ad7cde5a12f3aac46439ad0da68c0a/code/scripts/calc_round_numbers.py) or the `neptune` (https://github.com/filecoin-project/neptune) Rust library. Both methods calculate the round numbers via brute-force; by iterating over all reasonable values for R_F and R_P and choosing the pair that satisfies the security inequalities (see below) while minimizing the number of S-boxes.

$\textbf{const}\ (R_F, R_P) = \texttt{calc\_round\_numbers}(p, M, t, \alpha)$
The number of full and partial rounds; both are positive integers $R_F, R_P : \mathbb{Z}_{>0}$.

$\textbf{const}\ R = R_F + R_P$
The total number of rounds.

$\textbf{const}\ R_f = \lfloor R_F / 2 \rfloor$
Half the number of full rounds.

Security Inequalities

The round numbers (R_F, R_P) are chosen such that they satisfy the following inequalities:

$(1) \quad 2^M \le p^t \iff M \le t \log_2(p)$
The security level must be less than the bit-length of Poseidon's internal state vector $\mathbb{Z}_p^{[t]}$ (taken from Appendix C.1.1 in the Poseidon Paper (https://eprint.iacr.org/2019/458.pdf)). This is always satisfied for Filecoin's choice of p and M.

$(2) \quad R_f \ge 6$
The minimum R_F necessary to prevent statistical attacks (Eq. 2 Section 5.5.1 in the Poseidon Paper (https://eprint.iacr.org/2019/458.pdf) where $\lfloor \log_2(p) \rfloor - 2 = 252$ and $\mathcal{C} = 2$ for $\alpha = 5$).

$$(3) \quad R > \lceil M \log_\alpha(2) \rceil + \lceil \log_\alpha(t) \rceil \implies R > \begin{cases} 57 & \text{if } t \in [2, 5] \\ 58 & \text{if } t \in [6, 25] \end{cases}$$
The minimum number of total rounds necessary to prevent interpolation attacks (Eq. 3 Section 5.5.2 of the Poseidon Paper (https://eprint.iacr.org/2019/458.pdf)).

$$(4a) \quad R > \tfrac{M \log_\alpha(2)}{3} \implies R > 18.3$$
$$(4b) \quad R > t - 1 + \tfrac{M \log_\alpha(2)}{t+1}$$
The minimum number of total rounds required to prevent against Gaussian elimination attacks (both equations must be satisfied, Eq. 5 from Section 5.5.2 of the Poseidon Paper (https://eprint.iacr.org/2019/458.pdf)).

Round Constants

For each distinct Poseidon instance an array RoundConstants of Rt field elements is generated (t field elements per round) using the Grain-LFSR stream cipher whose 80-bit state is initialized to an encoding of the Poseidon instance.

$$\textbf{const}\ \text{FieldBits} : \{0,1\}^{[2]}_\text{msb} = \begin{cases} 0 & \text{if using a binary field } \mathbb{Z}_{2^m} \\ 1 & \text{if using a prime field } \mathbb{Z}_p \end{cases} = 01_2$$
Specifies the field type as prime or binary. Filecoin uses a prime field $\mathbb{Z}_p$, however the Poseidon paper allows Poseidon to be instantiated over binary fields $\mathbb{Z}_{2^m}$.

$$\textbf{const}\ \text{SboxBits} : \{0,1\}^{[4]}_\text{msb} = \begin{cases} 0 & \text{if } \alpha = 3 \\ 1 & \text{if } \alpha = 5 \\ 2 & \text{if } \alpha = \text{-1} \end{cases} = 0001_2$$
Specifies the S-box exponent α. Filecoin uses $\alpha = 5$.

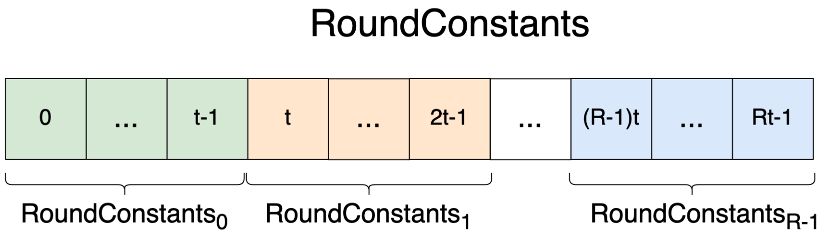
$\textbf{const}\ \text{FieldSizeBits} : \{0,1\}^{[12]}_\text{msb} = \lceil \log_2(p) \rceil = 255 = 000011111111_2$
The bit-length of the field modulus.

$$\begin{aligned} &\textbf{const}\ \text{GrainState}_\text{init} : \{0,1\}^{[80]} = \\ &\quad \text{FieldBits} \\ &\quad \|\ \text{SboxBits} \\ &\quad \|\ \text{FieldSizeBits} \\ &\quad \|\ t\ \textbf{as}\ \{0,1\}^{[12]}_\text{msb} \\ &\quad \|\ R_F\ \textbf{as}\ \{0,1\}^{[10]}_\text{msb} \\ &\quad \|\ R_P\ \textbf{as}\ \{0,1\}^{[10]}_\text{msb} \\ &\quad \|\ 1^{[30]} \end{aligned}$$
Initializes the Grain LFSR stream cipher which is used to derive RoundConstants for a Poseidon instance (p, M, t).

const $\text{RoundConstants} : \mathbb{Z}_p^{[Rt]}$
The round constants for a Poseidon instance (p, M, t).

Algorithm: RoundConstants

```
1.  $\text{state} : \{0,1\}^{[80]} = \text{GrainState}_{\text{init}}$
2.  **do** 160 **times:**
3.      $\text{bit} : \{0,1\} = \bigoplus_{\text{xor } i \in \{0,13,23,38,51,62\}} \text{state}[i]$
4.      $\text{state} = \text{state}[1..] \,\|\, \text{bit}$
5.  $\text{RoundConstants} : \mathbb{Z}_p^{[Rt]} = [\,]$
6.  **while len**$(\text{RoundConstants}) < Rt$**:**
7.      $\text{bits} : \{0,1\}^{[255]} = [\,]$
8.      **while len**$(\text{bits}) < 255$**:**
9.          $\text{bit}_1 = \bigoplus_{\text{xor } i \in \{0,13,23,38,51,62\}} \text{state}[i]$
10.         $\text{state} = \text{state}[1..] \,\|\, \text{bit}_1$
11.         $\text{bit}_2 = \bigoplus_{\text{xor } i \in \{0,13,23,38,51,62\}} \text{state}[i]$
12.         $\text{state} = \text{state}[1..] \,\|\, \text{bit}_2$
13.         **if** $\text{bit}_1 = 1$**:**
14.             $\text{bits}.\textbf{push}(\text{bit}_2)$
15.     $c = \text{bits}_{\text{msb}}$ **as** $\mathbb{Z}_{2^{255}}$
16.     if $c \in \mathbb{Z}_p$**:**
17.         $\text{RoundConstants}.\textbf{push}(c)$
18. **return** RoundConstants
```



const $\text{RoundConstants}_r : \mathbb{Z}_p^{[t]} = \text{RoundConstants}[rt..(r+1)t]$
Denotes the round constants for round $r \in [R]$ for an unoptimized Poseidon instance.

MDS Matrix

const $\mathbf{x} : \mathbb{Z}_p^{[t]} = [0, \ldots, t-1]$
const $\mathbf{y} : \mathbb{Z}_p^{[t]} = [t, \ldots, 2t-1]$

$$\textbf{const } \mathcal{M} : \mathbb{Z}_p^{[t \times t]} = \begin{bmatrix} (\mathbf{x}_0 + \mathbf{y}_0)^{-1} & \ldots & (\mathbf{x}_0 + \mathbf{y}_{t-1})^{-1} \\ \vdots & \ddots & \vdots \\ (\mathbf{x}_{t-1} + \mathbf{y}_0)^{-1} & \ldots & (\mathbf{x}_{t-1} + \mathbf{y}_{t-1})^{-1} \end{bmatrix}$$

The MDS matrix $\mathcal{M}$ for a Poseidon instance of width t. The superscript $^{-1}$ denotes a multiplicative inverse $\bmod p$. The MDS matrix is invertible and symmetric.

Domain Separation

Every preimage hashed by a Poseidon instance is associated with a hash type HashType to specify how Poseidon is being used. Filecoin uses two hash types: one designating a preimage as being for a Merkle tree hash function of arity $t-1$ and a second designating a preimage as having length $\textbf{len}(\text{preimage}) < t$ and having no specific application.

The HashType for a preimage determines the first element of Poseidon's initial $\text{state}[0] = \text{DomainTag}$, and any padding Padding that is applied to Poseidon's initial state.

$\textbf{const}\ \text{HashType} \in \{\text{MerkleTree}, \text{ConstInputLen}\}$
The allowed hash types in which to hash a preimage for a Poseidon instance (p, M, t). It is required that $0 < \textbf{len}(\textsf{preimage}) < t$.

- A HashType of MerkleTree designates a preimage as being the preimage of a Merkle tree hash function, where the tree is $t{-}1{:}1$ (i.e. $\text{Arity} = \textbf{len}(\textsf{preimage})$ number of nodes are hashed into 1 node).
- A HashType of ConstInputLen designates Poseidon as being used to hash preimages of length exactly $\textbf{len}(\textsf{preimage})$ into a single output element (where $0 < \textbf{len}(\textsf{preimage}) < t$).

$$\textbf{const}\ \text{DomainTag} : \mathbb{Z}_p = \begin{cases} 2^{\text{Arity}} - 1 & \text{if HashType} = \text{MerkleTree} \\ 2^{64}\,\textbf{len}(\textsf{preimage}) & \text{if HashType} = \text{ConstInputLen} \end{cases}$$
The first element $\textsf{state}[0] = \text{DomainTag}$ of the initial Poseidon **state**.

$$\textbf{const}\ \text{Padding} : \mathbb{Z}_p{}^{[*]} = \begin{cases} [\,] & \text{if HashType} = \text{MerkleTree} \\ 0^{[t-1-\textbf{len}(\textsf{preimage})]} & \text{if HashType} = \text{ConstInputLen} \end{cases}$$
The padding that is applied to Poseidon's initial state. If $\text{HashType} = \text{MerkleTree}$, no padding is applied. Otherwise, if $\text{HashType} = \text{ConstInputLen}$, the last $t - 1 - \textbf{len}(\textsf{preimage})$ elements of Poseidon's initial **state** are set to zero.

Unoptimized v.s. Optimized

Filecoin's rust library `neptune` (https://github.com/filecoin-project/neptune) implements the Poseidon hash function. The library differentiates between unoptimized and optimized Poseidon using the terms *correct* and *static* respectively.

The primary differences between the two versions are:

- the unoptimized algorithm uses the round constants RoundConstants, performs round constant addition before S-boxes, and uses the MDS matrix $\mathcal{M}$ for mixing
- the optimized algorithm uses the transformed rounds constants $\text{RoundConstants}'$ (containing fewer constants than RoundConstants), performs a round constant addition before the first round's S-box, performs round constant addition after every S-box other than the last round's, and uses multiple matrices for MDS mixing $\mathcal{M}$, $\mathcal{P}$, and $\mathcal{S}$. This change in MDS mixing from a non-sparse matrix $\mathcal{M}$ to sparse matrices $\mathcal{S}$ greatly reduces the number of multiplications in each round.

For a given Poseidon instance (p, M, t) the optimized and unoptimized algorithms will produce the same output when provided with the same input.

Unoptimized Poseidon

The Posiedon hash function takes a preimage of $t - 1$ prime field $\mathbb{Z}_p$ elements to a single field element. Poseidon operates on an internal state **state** of t field elements which, in the unoptimized algorithm, are transformed over R number of rounds of: round constant addition, S-boxes, and MDS matrix mixing. Once all rounds have been performed, Poseidon outputs the second element of the state.

A Posiedon hash function is instantiated by a parameter triple (p, M, t) which sets the prime field, the security level, and the size of Poseidon's internal state buffer **state**. From (p, M, t) the remaining Poseidon parameters are computed $(\alpha, R_F, R_P, \text{RoundConstants}, \mathcal{M})$, i.e. the S-box exponent, the round numbers, the round constants, and the MDS matrix.

The S-box function is defined as:

$$S : \mathbb{Z}_p \to \mathbb{Z}_p$$
$$S(x) = x^\alpha$$

The **state** is initialized to the concatenation of the DomainTag, **preimage**, and Padding:

$$\textsf{state} = \text{DomainTag} \,\|\, \textsf{preimage} \,\|\, \text{Padding}$$

Every round $r \in [R]$ begins with t field additions of the **state** with that round's constants RoundConstants_r:

$$\textsf{state} = [\textsf{state}[i] \oplus \text{RoundConstants}_r[i]]_{i \in [t]}$$

If r is a full round, i.e. $r < R_f$ or $r \geq R_f + R_P$, the S-box function is applied to each element of **state**:

$$\textsf{state} = [\textsf{state}[i]^{\alpha}]_{i \in [t]}$$

otherwise, if r is a partial round $r \in [R_f, R_f + R_P)$, the S-box function is applied to just the first **state** element:

$$\textsf{state}[0] = \textsf{state}[0]^{\alpha}$$

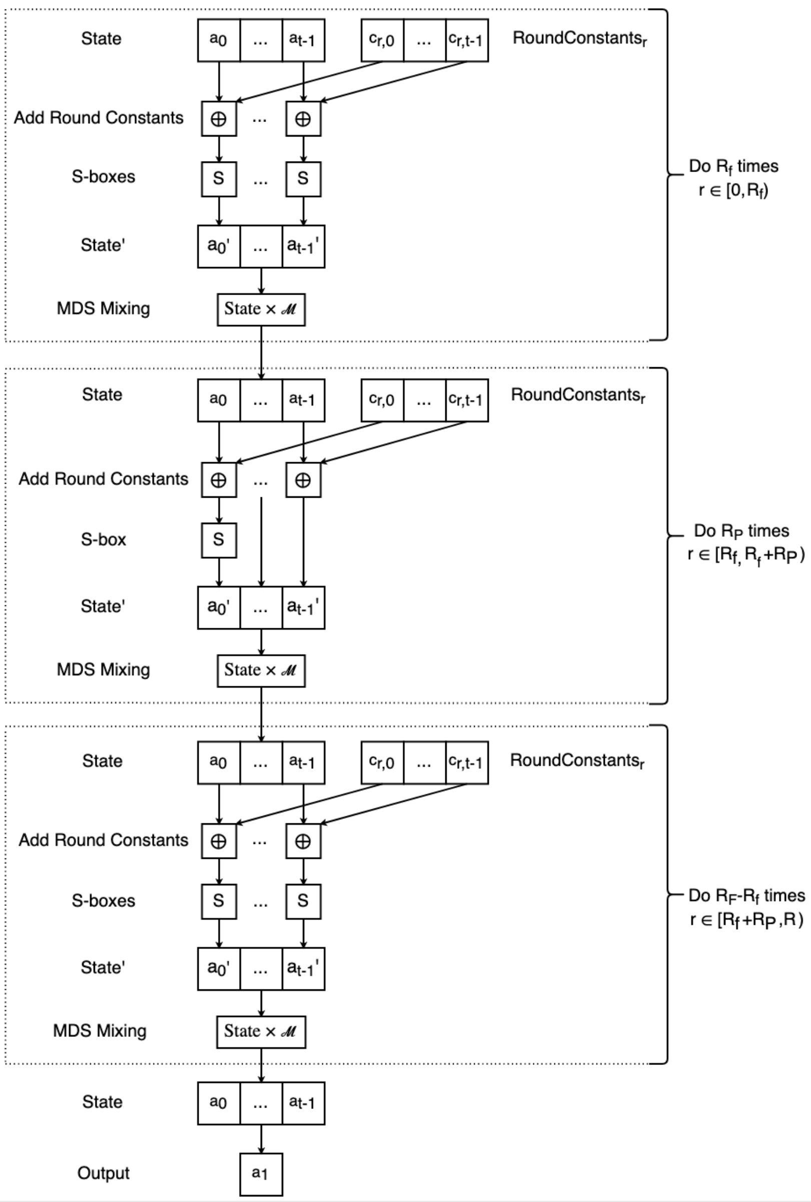
Once the S-boxes have been applied for a round, the **state** is transformed via vector-matrix multiplication with the MDS matrix $\mathcal{M}$:

$$\textsf{state} = \textsf{state} \times \mathcal{M}$$

After R rounds of the above procedure, Poseidon outputs the digest $\textsf{state}[1]$.

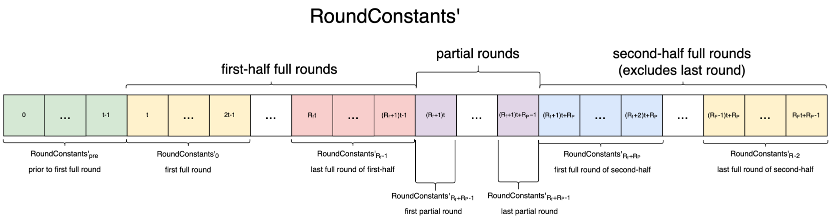
Function: $\textsf{poseidon}(\textsf{preimage} : {\mathbb{Z}_p}^{[t-1]}) \to \mathbb{Z}_p$

```
1.  $\textsf{state} : {\mathbb{Z}_p}^{[t]} = \text{DomainTag} \parallel \textsf{preimage} \parallel \text{Padding}$
2.  $\textbf{for } r \in [R_f]$:
3.      $\textsf{state} = [(\textsf{state}[i] \oplus \text{RoundConstants}_r[i])^5]_{i \in [t]} \times \mathcal{M}$
4.  $\textbf{for } r \in [R_f, R_f + R_P)$:
5.      $\textsf{state} = [\textsf{state}[i] \oplus \text{RoundConstants}_r[i]]_{i \in [t]}$
6.      $\textsf{state}[0] = \textsf{state}[0]^5$
7.      $\textsf{state} = \textsf{state} \times \mathcal{M}$
8.  $\textbf{for } r \in [R_f + R_P, R)$:
9.      $\textsf{state} = [(\textsf{state}[i] \oplus \text{RoundConstants}_r[i])^5]_{i \in [t]} \times \mathcal{M}$
10. $\textbf{return } \textsf{state}[1]$
```



Optimized Round Constants

Given the round constants RoundConstants and MDS matrix $\mathcal{M}$ for a Poseidon instance, we are able to derive round constants $\text{RoundConstants}'$ for the optimized Poseidon algorithm.


RoundConstants'
first-half full rounds
partial rounds
second-half full rounds (excludes last round)
0
…
t-1
t
2t-1
R_ft
$(R_f+1)t-1$
$(R_f+1)t$
$(R_f+1)t+R_P-1$
$(R_f+1)t+R_P$
$(R_f+2)t+R_P$
$(R_F-1)t+R_P$
R_Ft+R_P-1
$\text{RoundConstants}'_{pre}$
prior to first full round
$\text{RoundConstants}'_0$
first full round
$\text{RoundConstants}'_{R_f-1}$
last full round of first-half
$\text{RoundConstants}'_{R_f+R_P}$
first full round of second-half
$\text{RoundConstants}'_{R-2}$
last full round of second-half
$\text{RoundConstants}'_{R_f+R_P-1}$
first partial round
$\text{RoundConstants}'_{R_f+R_P-1}$
last partial round

$\textbf{const}\ \text{RoundConstants}' : \mathbb{Z}_p^{[tR_F+R_P]}$
The round constants for a Poseidon instance's (p, M, t) optimized hashing algorithm. Each full round is associated with t round constants, while each partial round is associated with one constant.

Algorithm: $\mathrm{RoundConstants}'$

```
1.  $\mathrm{RoundConstants}' : \mathbb{Z}_p{}^{[tR_F+R_P]} = [\,]$
2.  $\mathrm{RoundConstants}'$.**extend**($\mathrm{RoundConstants}_0$)
3.  **for** $r \in [1, R_f)$**:**
4.      $\mathrm{RoundConstants}'$.**extend**($\mathrm{RoundConstants}_r \times \mathcal{M}^{-1}$)
5.  partial_consts : $\mathbb{Z}_p{}^{[R_P]} = [\,]$
6.  acc : $\mathbb{Z}_p{}^{[t]} = \mathrm{RoundConstants}_{R_f+R_P}$
7.  **for** $r \in$ **reverse**($[R_f, R_f + R_P)$)**:**
8.      acc' $=$ acc $\times \mathcal{M}^{-1}$
9.      partial_consts.**push**(acc'$[0]$)
10.     acc'$[0] = 0$
11.     acc $=$ acc' $\vec{\oplus}\ \mathrm{RoundConstants}_r$
12. $\mathrm{RoundConstants}'$.**extend**(acc $\times \mathcal{M}^{-1}$)
13. $\mathrm{RoundConstants}'$.**extend**(**reverse**(partial_consts))
14. **for** $r \in [R_f + R_P + 1, R)$
15.     $\mathrm{RoundConstants}'$.**extend**($\mathrm{RoundConstants}_r \times \mathcal{M}^{-1}$)
16. **return** $\mathrm{RoundConstants}'$
```

Algorithm Comments:
Note: $\times$ denotes row-vector-matrix multiplication which outputs a row-vector.
Line 2. The first t round constants are unchanged. Note that both $\mathrm{RoundConstants}'_0$ and $\mathrm{RoundConstants}'_1$ are used in the first optimized round $r = 0$.
Lines 3-4. For each first-half full round, transform the round constants into $\mathrm{RoundConstants}_r \times \mathcal{M}^{-1}$.
Lines 5. Create a variable to store the round constants for the partial rounds **partial_consts** (in reverse order)
Lines 6. Create and initialize a variable **acc** that is transformed and added to $\mathrm{RoundConstants}_r$ in each **do** loop iteration.
Lines 7-11. For each partial round r (starting from the greatest partial round index $R_f + R_P - 1$ and proceeding to the least R_f) transform **acc** into **acc** $\times \mathcal{M}^{-1}$, take its first element as a partial round constant, then perform element-wise addition with $\mathrm{RoundConstants}_r$. The value of **acc** at the end of the i^{th} loop iteration is:
$\mathsf{acc}_i = \mathrm{RoundConstants}_r[0] \parallel ((\mathsf{acc}_{i-1} \times \mathcal{M}^{-1})[1..] \vec{\oplus}\ \mathrm{RoundConstants}_r[1..])$
Line 12. Set the last first-half full round's constants using the final value of **acc**.
Line 13. Set the partial round constants.
Lines 14-15. Set the remaining full round constants.

$\mathbf{const}\ \mathrm{RoundConstants}'_{\mathrm{pre}} : \mathbb{Z}_p{}^{[t]} = \mathrm{RoundConstants}'[..t]$
The first t constants in $\mathrm{RoundConstants}'$ are added to **state** prior to applying the S-box in the first round round $r = 0$.

$$\mathbf{const}\ \mathrm{RoundConstants}'_r : \mathbb{Z}_p{}^{[*]} = \begin{cases} \mathrm{RoundConstants}'[(r+1)t..(r+2)t] & \text{if } r \in \\ \mathrm{RoundConstants}'[(R_f+1)t + \delta] & \text{if } r \in \\ \mathrm{RoundConstants}'[(R_f+\delta+1)t + R_P..(R_f+\delta+2)t + R_P] & \text{if } r \in \end{cases}$$

For each round excluding the last $r \in [R-1]$, $\mathrm{RoundConstants}'_r$ is added to the Poseidon **state** after taht round's S-box has been applied.

Sparse MDS Matrices

A *sparse matrix* m is a square $n{\times}n$ matrix whose first row and first column are utilized and where all other entries are the $n{-}1{\times}n{-}1$ identity matrix $\mathcal{I}_{n-1}$:

$$m = \left[\begin{array}{c|c} m_{0,0} & m_{0,1..} \\ \hline m_{1..,0} & \mathcal{I}_{n-1} \end{array}\right] = \begin{bmatrix} m_{0,0} & \dots & \dots & m_{0,n-1} \\ \vdots & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & 0 & \dots & 1 \end{bmatrix}$$

The MDS matrix $\mathcal{M}$ is factored into a non-sparse matrix $\mathcal{P}$ and an array of sparse matrices $\mathcal{S} = [\mathcal{S}_0, \dots, \mathcal{S}_{R_P-1}]$ (one matrix per partial round). $\mathcal{P}$ is used in MDS mixing for the last first-half full round $r = R_f - 1$. Each matrix of $\mathcal{S}$ is used in MDS mixing for a partial round. $\mathcal{S}_0$ is used in the first partial round ($r = R_f$) and $\mathcal{S}_{R_P-1}$ is used in the last partial round ($r = R_f + i$).

const $\mathcal{P} : \mathbb{Z}_p{}^{[t\times t]}$
The *pre-sparse* matrix (a non-sparse matrix) used in MDS mixing for round for the last full round of the first-half $r = R_f - 1$. Like the MDS matrix $\mathcal{M}$, the pre-sparse matrix $\mathcal{P}$ is symmetric.

const $\mathcal{S} : \mathbb{Z}_p{}^{[t\times t][R_P]}$
The array of sparse matrices that $\mathcal{M}$ is factored into, which are used for MDS mixing in the optimized partial rounds.

Algorithm: $\mathcal{P}, \mathcal{S}$

```
1. sparse : $\mathbb{Z}_p{}^{[t\times t][R_P]} = [\,]$
2. $m : \mathbb{Z}_p{}^{[t\times t]} = \mathcal{M}$
3. do $R_P$ times:
4.     $(m', m'') : (\mathbb{Z}_p{}^{[t\times t]}, \mathbb{Z}_p{}^{[t\times t]}) = \textbf{sparse\_factorize}(m)$
5.     sparse.push$(m'')$
6.     $m = \mathcal{M} \times m'$
7. $\mathcal{P} = m$
8. $\mathcal{S} = \textbf{reverse}(\text{sparse})$
9. return $\mathcal{P}, \mathcal{S}$
```

Algorithm Comments:
Line 1. An array containing the sparse matrices that $\mathcal{M}$ is factored into.
Line 2. An array m that is repeatedly factored into a non-sparse matrix m' and a sparse matrix m'', i.e. $m = m' \times m''$.
Lines 3-6. In each loop iteration we factor m into m' and m''. The first **do** loop iteration calculates the sparse matrix m'' used in MDS mixing for last partial round $r = R_f + R_P - 1$. The last **do** loop iteration calculates the sparse matrix m'' used in MDS mixing for the first partial round $r = R_f$ (i.e. $\mathcal{S}_0 = m''$).
Line 6. $\mathcal{M} \times m'$ is a matrix-matrix multiplication which produces a $t{\times}t$ matrix.

The function **sparse_factorize** factors a non-sparse matrix m into a non-sparse matrix m' and sparse matrix m'' such that $m = m' \times m''$.

Function: sparse_factorize$(m : \mathbb{Z}_p{}^{[t\times t]}) \rightarrow (m' : \mathbb{Z}_p{}^{[t\times t]}, m'' : \mathbb{Z}_p{}^{[t\times t]})$

```
1. $\hat{m} : \mathbb{Z}_p{}^{[t-1\times t-1]} = m_{1..,1..} = \begin{bmatrix} m_{1,1} & \dots & m_{1,t-1} \\ \vdots & \ddots & \vdots \\ m_{t-1,1} & \dots & m_{t-1,t-1} \end{bmatrix}$
2. $m' : \mathbb{Z}_p{}^{[t\times t]} = \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \hat{m} \end{array}\right] = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & m_{1,1} & \dots & m_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & m_{t-1,1} & \dots & m_{t-1,t-1} \end{bmatrix}$
3. $\mathbf{w} : \mathbb{Z}_p{}^{[t-1\times 1]} = \hat{m}_{*,0} = \begin{bmatrix} \hat{m}_{0,0} \\ \vdots \\ \hat{m}_{t-2,0} \end{bmatrix} = m_{1..,1} = \begin{bmatrix} m_{1,1} \\ \vdots \\ m_{t-1,1} \end{bmatrix}$
4. $\hat{\mathbf{w}} : \mathbb{Z}_p{}^{[t-1\times 1]} = \hat{m}^{-1} \times \mathbf{w} = \begin{bmatrix} \hat{m}^{-1}{}_{0,*} \cdot \mathbf{w} \\ \vdots \\ \hat{m}^{-1}{}_{t-2,*} \cdot \mathbf{w} \end{bmatrix}$
5. $m'' : \mathbb{Z}_p{}^{[t\times t]} = \left[\begin{array}{c|c} m_{0,0} & m_{0,1..} \\ \hline \hat{\mathbf{w}} & \mathcal{I}_{t-1} \end{array}\right] = \begin{bmatrix} m_{0,0} & \dots & \dots & m_{0,t-1} \\ \hat{\mathbf{w}}_0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{w}}_{t-2} & 0 & \dots & 1 \end{bmatrix}$
6. return $m', m''$
```

Algorithm Comments:
Line 1. $\hat{m}$ is a submatrix of m which excludes m's first row and first column.
Line 2. m' is a copy of m where m's first row and first column have been replaced with $[1, 0, \dots, 0]$.
Line 3. $\mathbf{w}$ is a column vector which is the first column of $\hat{m}$ or the second column of m excluding the first row's value.
Line 4. $\hat{\mathbf{w}}$ is the matrix-column vector product of $\hat{m}^{-1}$ and $\mathbf{w}$.

Line 5. m'' is a sparse matrix whose first row is the first row of m, remaining first column is $\hat{\mathbf{w}}$, and remaining entries are the identity matrix.

Optimized Poseidon

The optimized Poseidon hash function is instantiated in the same way as the unoptimized algorithm, however the optimized Poseidon algorithm requires the additional precomputation of round constants $\text{RoundConstants}'$, a *pre-sparse* matrix $\mathcal{P}$, and sparse matrices $\mathcal{S}$.

Prior to the first round $r = 0$, the state is initialized and added to the *pre-* round constants $\text{RoundConstants}'_{\text{pre}}$:

$$\begin{aligned} \textsf{state} &= \text{DomainTag} \parallel \textsf{preimage} \parallel \text{Padding} \\ \textsf{state} &= \textsf{state} \vec{\oplus} \text{RoundConstants}'_{\text{pre}} \end{aligned}$$

For each full round of the first-half $r \in [R_f]$: the S-box function is applied to each element of **state**, the output of each S-box is added to the associated round constant in $\text{RoundConstants}'_r$, and MDS mixing occurs between **state** and the MDS matrix $\mathcal{M}$ or the *pre-sparse* matrix $\mathcal{P}$ (when $r < R_f - 1$ and $r = R_f - 1$ respectively).

$$\textsf{state} = \begin{cases} [\textsf{state}[i]^\alpha \oplus \text{RoundConstants}'_r[i]] \times \mathcal{M} & \text{if } r \in [R_f - 1] \\ [\textsf{state}[i]^\alpha \oplus \text{RoundConstants}'_r[i]] \times \mathcal{P} & \text{if } r = R_f - 1 \end{cases}$$

For each partial round $r \in [R_f, R_f + R_P)$ the S-box function is applied to the first **state** element, the round constant is added to the first **state** element, and MDS mixing occurs between the **state** and the i^{th} sparse matrix $\mathcal{S}_i$ (the i^{th} partial round $i \in [R_P]$ is associated with sparse matrix $\mathcal{S}_i$ where $i = r - R_f$):

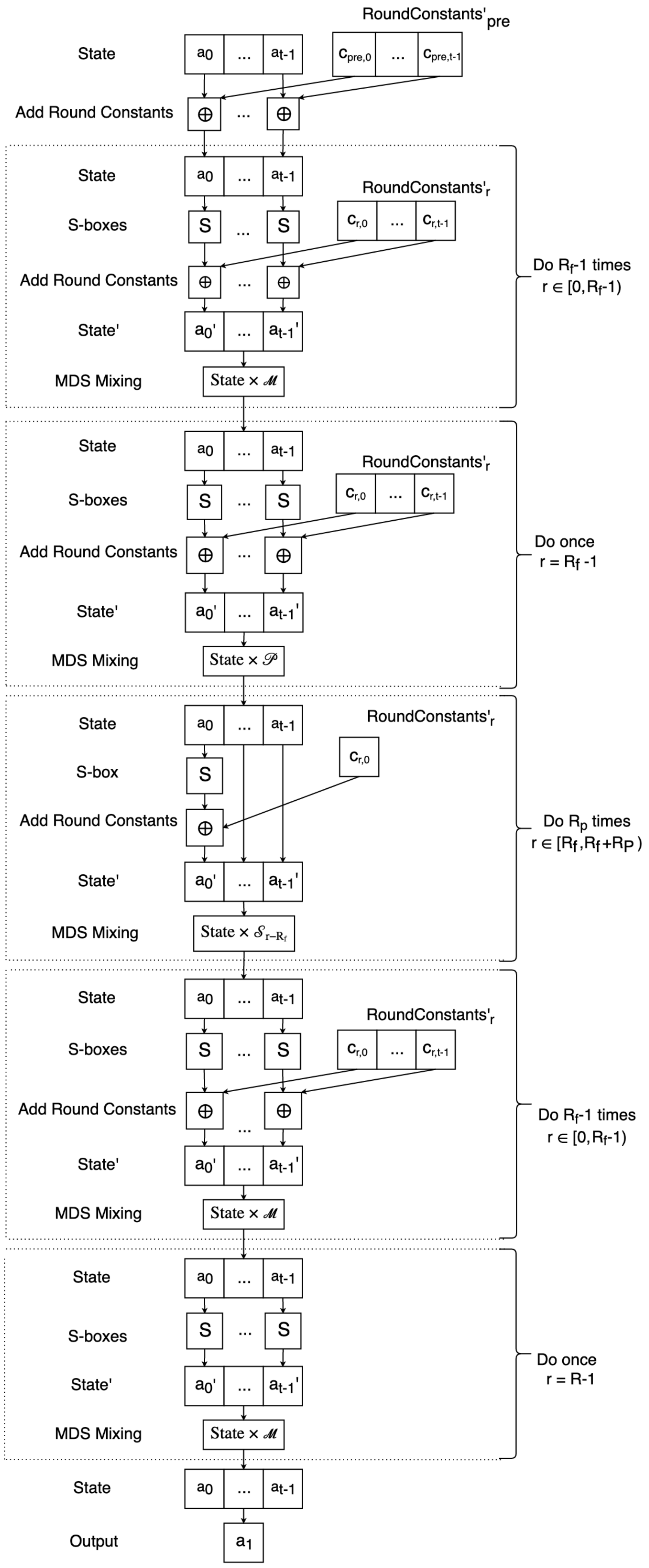
$$\begin{aligned} \textsf{state}[0] &= \textsf{state}[0]^\alpha \oplus \text{RoundConstants}'_r \\ \textsf{state} &= \textsf{state} \times \mathcal{S}_{r-R_f} \end{aligned}$$

The second half of full rounds $r \in [R_f + R_P, R)$ proceed in the same way as the first half of full rounds except that all MDS mixing uses the MDS matrix $\mathcal{M}$ and that the last round $r = R - 1$ does not add round constants into **state**.

After performing R rounds, Poseidon outputs the digest $\textsf{state}[1]$.

```
Function: poseidon($\textsf{preimage} : \mathbb{Z}_p^{[t-1]}$) $\to \mathbb{Z}_p$
1.  $\textsf{state} : \mathbb{Z}_p^{[t]} = \text{DomainTag} \parallel \textsf{preimage} \parallel \text{Padding}$
2.  $\textsf{state} = [\textsf{state}[i] \oplus \text{RoundConstants}'_{\text{pre}}[i]]_{i \in [t]}$
3.  for $r \in [R_f - 1]$:
4.      $\textsf{state} = [\textsf{state}[i]^5 \oplus \text{RoundConstants}'_r[i]]_{i \in [t]} \times \mathcal{M}$
5.  $\textsf{state} = [\textsf{state}[i]^5 \oplus \text{RoundConstants}'_{R_f - 1}[i]]_{i \in [t]} \times \mathcal{P}$
6.  for $r \in [R_f, R_f + R_P)$:
8.      $\textsf{state}[0] = \textsf{state}[0]^5 \oplus \text{RoundConstants}'_r$
9.      $\textsf{state} = \textsf{state} \times \mathcal{S}_{r-R_f}$
10. for $r \in [R_f + R_P, R - 1)$:
11.     $\textsf{state} = [\textsf{state}[i]^5 \oplus \text{RoundConstants}'_r[i]]_{i \in [t]} \times \mathcal{M}$
12. $\textsf{state} = [\textsf{state}[i]^5]_{i \in [t]} \times \mathcal{M}$
13. return $\textsf{state}[1]$
```

Algorithm Comments:
Line 2. Add pre- $r{=}0$ round constants.
Line 3-4. Perform first-half full rounds up to and excluding the last (mixing is done using $\mathcal{M}$).
Line 5. Perform the last first-half full round $r{=}R_f - 1$ (mixing is done using the *pre-sparse* matrix $\mathcal{P}$).
Lines 6-9. Perform the partial rounds (mixing is done using each round r's sparse matrix $\mathcal{S}_{r-R_f}$).
Lines 10-11. Perform the second-half full rounds excluding the last round (mixing is done using $\mathcal{M}$).
Line 12. Perform the last round where no round constants are added (mixing is done using $\mathcal{M}$).


RoundConstants'pre
State
a_0 … a_{t-1}
$c_{pre,0}$ … $c_{pre,t-1}$
Add Round Constants
State
S-boxes
RoundConstants'r
$c_{r,0}$ … $c_{r,t-1}$
Add Round Constants
State'
a_0' … a_{t-1}'
MDS Mixing
State × $\mathscr{M}$
Do R_f-1 times $r \in [0, R_f\text{-}1)$
State
S-boxes
RoundConstants'r
Add Round Constants
State'
MDS Mixing
State × $\mathscr{P}$
Do once $r = R_f$ -1
State
RoundConstants'r
$c_{r,0}$
S-box
Add Round Constants
State'
MDS Mixing
State × $\mathscr{S}_{r-R_f}$
Do R_p times $r \in [R_f, R_f + R_P)$
State
S-boxes
RoundConstants'r
Add Round Constants
State'
MDS Mixing
State × $\mathscr{M}$
Do R_f-1 times $r \in [0, R_f\text{-}1)$
State
S-boxes
State'
MDS Mixing
State × $\mathscr{M}$
Do once r = R-1
State
Output
a_1