Poseidon

General Notation

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

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

$A : \mathbb{T}^{[n \times n]}$
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.

$v : \mathbb{T}^{[n \times n]\,[m]}$
An array of m matrices.

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

$\mathsf{bits} : \{0, 1\}^{[n]}$
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 $0, \ldots, n-1$. Note that $[n] = [0, n)$.

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

Array Operations

Note: 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. $i \in \{1, 2, 3\}$).

$v \vec{\oplus} w$
Element-wise field addition of two equally lengthed vectors of field elements $v, w : {\mathbb{Z}_p}^{[n]}$. The above is equivalent to writing $[v_0 \oplus w_0, \ldots, v_{n-1} \oplus w_{n-1}]$.

$\mathbf{reverse}(v)$
Reverses the elements of a vector v, i.e. the first element of $\mathbf{reverse}(v)$ is the last element of v.

Matrix Operations

$A_{i,j}$
Returns the value of matrix A at row i column j.

$A_{i,*}$
Returns the i^{th} row of matrix A.

$A_{*,j}$
Returns the j^{th} column of matrix A.

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

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

$A \times B$
Matrix-matrix multiplication of two matrices of field elements $A : {\mathbb{Z}_p}^{[m \times n]}$ and $B : {\mathbb{Z}_p}^{[n \times k]}$ which produces a matrix of type ${\mathbb{Z}_p}^{[m \times k]}$. Note that $(A \times B)_{i,j} = A_{i,*} \cdot B_{*,j}$ where the dot product uses field multiplication.

A^{-1}
The inverse of a square $n \times n$ matrix, i.e. returns the matrix such that $A \times A^{-1} = \mathcal{I}_n$

$$\mathbf{v} \times A = [\mathbf{v}_0, \ldots, \mathbf{v}_{m-1}] \begin{bmatrix} A_{0,0} & \ldots & A_{0,n-1} \\ \vdots & \ddots & \vdots \\ A_{m-1,0} & \ldots & A_{m-1,n-1} \end{bmatrix} = [\mathbf{v} \cdot A_{*,i}]_{i \in [n]}$$

Vector-matrix multiplication of a row vector of field elements $\mathbf{v} : {\mathbb{Z}_p}^{[m]}$ with a matrix of field elements $m : {\mathbb{Z}_p}^{[m \times n]}$, note that $\mathbf{len}(\mathbf{v}) = \mathbf{rows}(A)$ and $\mathbf{len}(\mathbf{v} \times A) = \mathbf{columns}(A)$. The product is a row vector of length n (the number of matrix columns). The i^{th} element of the product vector is the dot product of $\mathbf{v}$ and the i^{th} column of A. Note that dot products use field multiplication.

$$A \times \mathbf{v} = \begin{bmatrix} A_{0,0} & \ldots & A_{0,n-1} \\ \vdots & \ddots & \vdots \\ A_{m-1,0} & \ldots & A_{m-1,n-1} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \vdots \\ \mathbf{v}_{n-1} \end{bmatrix} = \begin{bmatrix} A_{0,*} \cdot \mathbf{v} \\ \vdots \\ A_{m-1,*} \cdot \mathbf{v} \end{bmatrix}$$

Matrix-vector multiplication of a volumn vector $v : \mathbb{T}^{[m \times 1]}$ and matrix $A : \mathbb{T}^{[m \times n]}$, note that $\mathbf{rows}(\mathbf{v}) = \mathbf{columns}(A)$ and $\mathbf{rows}(A \times \mathbf{v}) = \mathbf{rows}(A)$. The product is a column vector whose length is equal to the number of rows of A. The i^{th} element of the product vector is the dot product of the i^{th} row of A with $\mathbf{v}$. Note that dot products use field multiplication.

Note: $\mathbf{v} \times A = (A \times \mathbf{v})^T$ when A is symmetric $A = A^T$ (the i^{th} row of A equals the i^{th} column of A), i.e. the row vector-matrix product and matrix-column vector product contain the same elements when A 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 an integer power α.

Bitwise Operations

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

$\bigoplus_{\text{xor } i \in \{1,2,3\}} i$
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

Note: the leftmost digit of an integer $x : \mathbb{Z}_{p \geq 0}$ is the most significant, thus a right-shift by n bits is defined: $x \gg n = x/2^n$.

x **as** $\{0,1\}^{[n]}_{\mathsf{msb}}$
Converts an integer $x : \mathbb{Z}_{\geq 0}$ into its n-bit binary representation. The most-significant bit (msb) is first (leftmost) in the produced bit array $\{0,1\}^{[n]}$. The above is equivalent to writing $\mathbf{reverse}([(x \gg i) \wedge 1]_{i \in [\lceil \log_2(x) \rceil]})$. For example, 6 **as** $\{0,1\}^{[3]}_{\mathsf{msb}} = [1, 1, 0]$.

$\mathsf{bits}_{\mathsf{msb}}$ **as** $\mathbb{Z}_{\geq 0}$
Converts a bit array $\mathsf{bits} : \{0,1\}^{[n]}$ into a unsigned (non-negative) integer where the first bit in bits is the most significant (msb). The above is equivalent to writing $\sum_{i \in [n]} 2^i * \mathbf{reverse}(\mathsf{bits})[i]$.

Poseidon-Specific Symbols

$p : \mathbb{Z}_{>0}$
The prime field modulus.

$M \in \{80, 128, 256\}$
The security level measured in bits. Poseidon allows for 80, 128, and 256-bit security levels.

$t : \mathbb{Z}_{>0} = \mathbf{len}(\mathsf{preimage}) + \mathbf{len}(\mathrm{digest}) = \mathbf{len}(\mathsf{preimage}) + \left\lceil \frac{2M}{\log_2(p)} \right\rceil$
The *width* of a Poseidon instance; the length in field elements of an instance's internal state array. The width t is equal to the preimage length plus the output length, where output length is equal to the number of field elements $\left\lceil \frac{2M}{\log_2(p)} \right\rceil$ required to achieve the targeted security level M in a field of size $\log_2(p)$. Stated another way, each field element in a Poseidon digest provides and additional $\frac{\log_2(p)}{2}$ bits of security.

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

$\alpha \in \{-1, 3, 5\}$
The S-box function's exponent $S(x) = x^\alpha$, where $\gcd(\alpha, p-1) = 1$. Poseidon allows for exponents -1, 3, and 5.

$R_F : \mathbb{Z}_{>0}$
The number of full rounds. R_F is even.

$R_P : \mathbb{Z}_{>0}$
The number of partial rounds.

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

$R_f = R_F/2$
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.

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

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

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

$\mathrm{RoundConstants} : \mathbb{Z}_p{}^{[Rt]}$
The round constants for an unoptimized Poseidon instance.

$\mathrm{RoundConstants}_r : \mathbb{Z}_p{}^{[t]}$
The round constants for round $r \in [R]$ for an unoptimized Poseidon instance, that are added to state before round r's S-boxes.

$\text{RoundConstants}' : \mathbb{Z}_p{}^{[tR_F+R_P]} = \text{RoundConstants}'_{\text{pre}} \parallel \text{RoundConstants}'_1 \parallel \ldots \parallel \text{RoundConstants}'_{R-2}$
The round constants for an optimized Poseidon instance. There are no constants associated with the last full round $r = R - 1$.

$\text{RoundConstants}'_{\text{pre}} : \mathbb{Z}_p{}^{[t]}$
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.

$$\text{RoundConstants}'_r : \begin{cases} \mathbb{Z}_p{}^{[1]} & \text{if } r \in [R_f, R_f + R_P) \quad \text{i.e. } r \text{ is a partial round} \\ \mathbb{Z}_p{}^{[t]} & \text{if } r \in [R_f] \text{ or } r \in [R_f + R_P, R - 1) \quad \text{i.e. } r \text{ is a full round, excluding the last round} \end{cases}$$
The round constants that are added to Poseidon's **state** array after the S-boxes in round r in an optimized Poseidon instance. Partial rounds have a single round constant, full rounds (excluding the last) have t constants. The last full round has no round constants.

$\mathcal{M} : \mathbb{Z}_p{}^{[t \times t]}$
The MDS matrix for a Poseidon instance.

$\mathcal{P} : \mathbb{Z}_p{}^{[t \times t]}$
The *pre-sparse* matrix used in MDS mixing for the last first-half full round ($r = R_f - 1$) of an optimized Poseidon instance.

$\mathcal{S} : \mathbb{Z}_p{}^{[t \times t]^{[R_P]}}$
An array of sparse matrices used in MDS mixing for the partial rounds $r \in [R_f, R_f + R_P)$ of the optimized Poseidon algorithm.

Poseidon 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 $\textbf{len}(\text{preimage}) \in [1, t)$.

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

Filecoin's Poseidon Instances

Note: the following are the Poseidon instantiation parameters used within Filecoin and do not represent all possible Poseidon instances.

$p = 52435875175126190479447740508185965837690552500527637822603658699938581184513$
$= 0\text{x}73\text{eda}753299\text{d}7\text{d}483339\text{d}80809\text{a}1\text{d}80553\text{bda}402\text{fffe}5\text{bfeffffffff}00000001$
The Poseidon prime field modulus in base-10 and base-16. Filecoin uses BLS12-381's scalar field as the Poseidon prime field $\mathbb{Z}_p$, i.e. p is the order of BLS12-381's prime order subgroup $\mathbb{G}_1$. The bit-length of p is $\lceil \log_2(p) \rceil = 255 \approx 256$ bits.

$M = 128 \text{ Bits}$
Filecoin targets the 128-bit security level.

$t \in \{3, 5, 9, 12\} = \{\text{arity} + 1 \mid \text{arity} \in \{2, 4, 8, 11\}\}$
The size in field elements of Poseidon's internal state; equal to the preimage length (a Filecoin Merkle tree arity) plus 1 for the digest length (the number of field elements required to target the $M = 128$-bit security level via a 256-bit prime p). Filecoin's Poseidon instances take preimages of varying lengths (2, 4, 8, and 11 field elements) and always return one field element.

- $t = 3$ 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, 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).

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

R_F and R_P are calculated using either the Python script `calc_round_numbers.py` or the `neptune` Rust library, denoted `calc_round_numbers`. 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 (provided below) while minimizing the number of S-boxes.

Security Inequalities

The round numbers R_F and R_P are chosen such that they satisfy the following inequalities. The symbol $\implies$ is used to indicate that an inequality simplifies when Filecoin's Poseidon parameters $M = 128$ and $\alpha = 5$ are plugged in.

(1) $2^M \leq p^t$
Equivalent to writing $M \leq t \log_2(p)$ (Appendix C.1.1 in the Poseidon paper). This is always satisfied for Filecoin's choice of p and M.

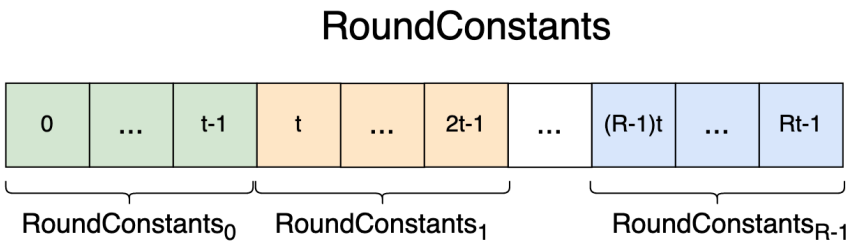
(2) $R_f \geq 6$
The minimum R_F necessary to prevent statistical attacks (Eq. 2 Section 5.5.1 in the Poseidon paper where $\lfloor \log_2(p) \rfloor - 2 = 252$ and $\mathcal{C} = 2$ for $\alpha = 5$).

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

(4a) $R > \frac{M \log_\alpha(2)}{3} \implies R > 18.3$
(4b) $R > t - 1 + \frac{M \log_\alpha(2)}{t+1}$
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).

Round Constants

Note: this section gives the round constants for only the unoptimized Poseidon algorithm.



const $\text{RoundConstants} : \mathbb{Z}_p{}^{[Rt]}$
For each Poseidon instance (p, M, t) an array RoundConstants containing Rt field elements is generated (t field elements per round $r \in [R]$) using the Grain-LFSR stream cipher whose 80-bit state is initialized to $\text{GrainState}_{\text{init}}$, an encoding of the Poseidon instance.

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

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 always uses a prime field $\mathbb{Z}_p$, however Poseidon also can be instantiated using a binary field $\mathbb{Z}_{2^m}$.

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 $\alpha = 5$.

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

$\textbf{const}\ \mathrm{GrainState}_{\mathrm{init}} : \{0,1\}^{[80]} =$
$\quad \mathrm{FieldBits}$
$\quad \| \ \mathrm{SboxBits}$
$\quad \| \ \mathrm{FieldSizeBits}$
$\quad \| \ t\ \textbf{as}\ \{0,1\}^{[12]}_{\mathsf{msb}}$
$\quad \| \ R_F\ \textbf{as}\ \{0,1\}^{[10]}_{\mathsf{msb}}$
$\quad \| \ R_P\ \textbf{as}\ \{0,1\}^{[10]}_{\mathsf{msb}}$
$\quad \| \ 1^{[30]}$

Initializes the Grain-LFSR stream cipher which is used to derive $\mathrm{RoundConstants}$ for a Poseidon instance (p, M, t).

Algorithm: $\mathrm{RoundConstants}$

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

MDS Matrix

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

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

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 is associated with a hash type $\mathrm{HashType}$ to encode the Poseidon application, note that $\mathrm{HashType}$ is specified per preimage and does not specify a Poseidon instance.

Filecoin uses two hash types $\mathrm{MerkleTree}$ and $\mathrm{ConstInputLen}$ to designate a preimage as being for a Merkle tree of arity $t-1$ or being for no specific application, but having a length $\textbf{len}(\mathsf{preimage}) < t$.

The $\mathrm{HashType}$ determines the $\mathrm{DomainTag}$ and $\mathrm{Padding}$ used for a preimage, which give the first element of Poseidon's initial state:

$$\mathsf{state} = \mathrm{DomainTag} \ \| \ \mathsf{preimage} \ \| \ \mathrm{Padding} \quad .$$

$\textbf{const}\ \mathrm{HashType} \in \{\mathrm{MerkleTree}, \mathrm{ConstInputLen}\}$
The allowed hash types in which to hash a preimage for a Poseidon instance (p, M, t). It is required that $1 \leq \textbf{len}(\mathsf{preimage}) < t$.

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

$$\textbf{const}\ \mathrm{DomainTag} : \mathbb{Z}_p = \begin{cases} 2^{\mathrm{arity}} - 1 & \text{if } \mathrm{HashType} = \mathrm{MerkleTree} \\ 2^{64} * \textbf{len}(\mathsf{preimage}) & \text{if } \mathrm{HashType} = \mathrm{ConstInputLen} \end{cases}$$

Encodes the Poseidon application within the first Poseidon initial state element $\mathsf{state}[0]$ for a preimage.

$\textbf{const}\ \text{Padding} : \mathbb{Z}_p{}^{[*]} = \begin{cases} [\,] & \text{if HashType} = \text{MerkleTree} \\ 0^{[t-1-\textbf{len}(\text{preimage})]} & \text{if HashType} = \text{ConstInputLen} \end{cases}$

The padding that is applied to Poseidon's initial state. A HashType of MerkleTree results in no applied padding; a HashType of ConstInputLen pads the last $t - 1 - \textbf{len}(\text{preimage})$ elements of Poseidon's initial state to zero.

Poseidon Hash Function

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, $\mathsf{preimage}$, and Padding:

$$\mathsf{state} = \text{DomainTag} \parallel \mathsf{preimage} \parallel \text{Padding} \quad .$$

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

$$\mathsf{state} = \mathsf{state} \vec{\oplus} \text{RoundConstants}_r \quad .$$

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:

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

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

$$\mathsf{state}[0] = \mathsf{state}[0]^\alpha \quad .$$

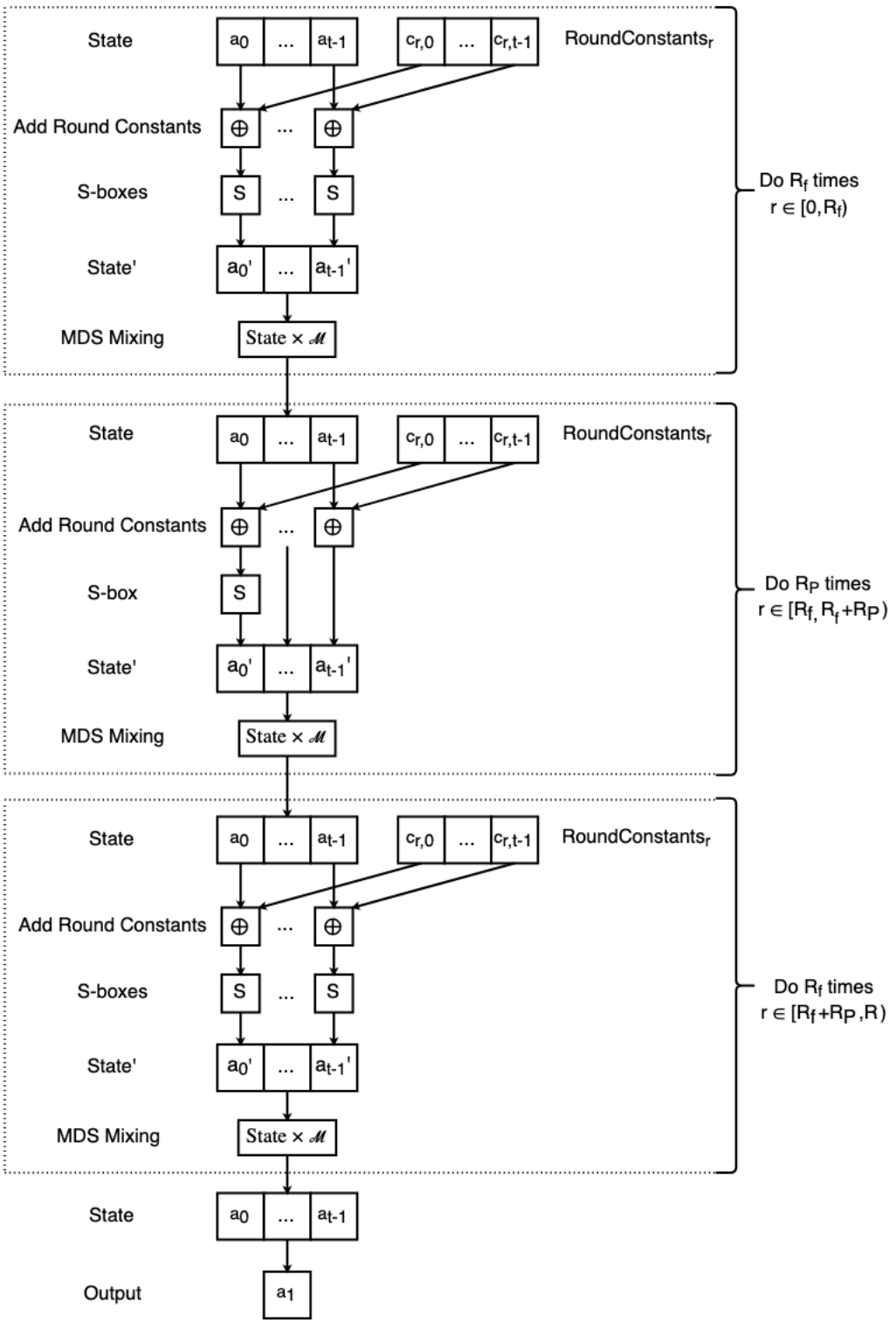
Once the S-boxes have been applied for a round, the state is transformed via vector-matrix multiplication with the MDS matrix:

$$\mathsf{state} = \mathsf{state} \times \mathcal{M} \quad .$$

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

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

1. $\mathsf{state} : \mathbb{Z}_p{}^{[t]} = \text{DomainTag} \parallel \mathsf{preimage} \parallel \text{Padding}$
2. **for** $r \in [R]$**:**
3. $\quad \mathsf{state} = \mathsf{state} \vec{\oplus} \text{RoundConstants}_r$
4. $\quad$ **if** $r \in [R_f]$ **or** $r \in [R_f + R_P, R)$**:**
5. $\quad\quad \mathsf{state} = [\mathsf{state}[i]^\alpha]_{i \in [t]}$
6. $\quad$ **else**
7. $\quad\quad \mathsf{state}[0] = \mathsf{state}[0]^\alpha$
8. $\quad \mathsf{state} = \mathsf{state} \times \mathcal{M}$
9. **return** $\mathsf{state}[1]$



Optimizations

Filecoin's rust library `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:

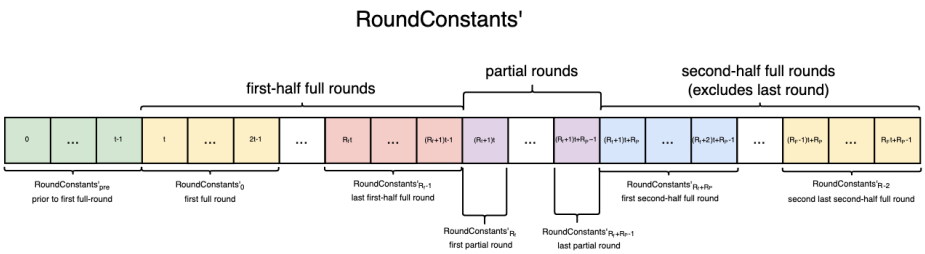
- the unoptimized algorithm uses the round constants $\mathrm{RoundConstants}$, performs round constant addition before S-boxes, and uses the MDS matrix $\mathcal{M}$ for mixing
- the optimized algorithm uses the transformed rounds constants $\mathrm{RoundConstants}'$ (containing fewer constants than $\mathrm{RoundConstants}$), 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.

Optimized Round Constants

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



const $\mathrm{RoundConstants}' : \mathbb{Z}_p^{[tR_F+R_P]}$
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. $\quad \mathrm{RoundConstants}'$**.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. $\quad$ acc$' =$ acc $\times \mathcal{M}^{-1}$
9. $\quad$ partial_consts**.push**(acc$'[0]$)
10. $\quad$ acc$'[0] = 0$
11. $\quad$ 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. $\quad \mathrm{RoundConstants}'$**.extend**$(\mathrm{RoundConstants}_r \times \mathcal{M}^{-1})$
16. **return** $\mathrm{RoundConstants}'$

Algorithm Comments:
Note: $\times$ denotes a 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 $\mathrm{RoundConstants}_r \times \mathcal{M}^{-1}$.
Line 5. Create a variable to store the round constants for the partial rounds partial_consts (in reverse order).
Line 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:

$$\text{acc}_i = \mathrm{RoundConstants}_r[0] \parallel ((\text{acc}_{i-1} \times \mathcal{M}^{-1})[1..] \vec{\oplus} \mathrm{RoundConstants}_r[1..])$$

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.

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

$$\textbf{const } \text{RoundConstants}'_r : {\mathbb{Z}_p}^{[*]} = \begin{cases} \text{RoundConstants}'[(r+1)t..(r+2)t] & \text{if } r \in [R_f] \\ \text{RoundConstants}'[(R_f+1)t + r_P] & \text{if } r \in [R_f, R_f + R_P), \text{where } r_P = r - R_f \text{ is the partial round inde} \\ \text{RoundConstants}'[(r_F+1)t + R_P..(r_F+2)t + R_P] & \text{if } r \in [R_f + R_P, R-1), \text{where } r_F = r - R_P \text{ is the full round inde} \end{cases}$$

For each round excluding the last $r \in [R-1]$, $\text{RoundConstants}'_r$ is added to the Poseidon state after that 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 $\mathcal{I}_{n-1}$:

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

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. The first sparse matrix $\mathcal{S}_0$ is used in the first partial round ($r = R_f$) and the last sparse matrix $\mathcal{S}_{R_P-1}$ is used in the last partial round ($r = R_f + i$).

$\textbf{const } \mathcal{P} : {\mathbb{Z}_p}^{[t \times t]}$
The *pre-sparse* matrix (a non-sparse matrix) used in MDS mixing 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.

$\textbf{const } \mathcal{S} : {\mathbb{Z}_p}^{[t \times t]\,[R_P]}$
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. $\quad (m', m'') : ({\mathbb{Z}_p}^{[t \times t]}, {\mathbb{Z}_p}^{[t \times t]}) = \text{sparse\_factorize}(m)$
5. $\quad \text{sparse}.\textbf{push}(m'')$
6. $\quad m = \mathcal{M} \times m'$
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. $\mathbf{w} : \mathbb{Z}_p{}^{[t-1 \times 1]} = m_{1..,0} = \begin{bmatrix} m_{1,0} \\ \vdots \\ m_{t-1,0} \end{bmatrix}$
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. $\mathbf{return}\ m', m''$

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 whose values are the first column of m excluding m's first row.
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 $\mathrm{RoundConstants}'$, 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-first round constants $\mathrm{RoundConstants}'_{\mathrm{pre}}$:

$$\mathsf{state} = \mathrm{DomainTag} \parallel \mathsf{preimage} \parallel \mathrm{Padding}$$
$$\mathsf{state} = \mathsf{state} \vec{\oplus} \mathrm{RoundConstants}'_{\mathrm{pre}}$$

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 $\mathrm{RoundConstants}'_r$, and MDS mixing occurs between state and the MDS matrix $\mathcal{M}$ (when $r < R_f - 1$) or the *pre-sparse* matrix $\mathcal{P}$ (when $r = R_f - 1$).

$$\mathsf{state} = \begin{cases} [\mathsf{state}[i]^\alpha \oplus \mathrm{RoundConstants}'_r[i]]_{i \in [t]} \times \mathcal{M} & \text{if } r \in [R_f - 1] \\ [\mathsf{state}[i]^\alpha \oplus \mathrm{RoundConstants}'_r[i]]_{i \in [t]} \times \mathcal{P} & \text{if } r = R_f - 1 \end{cases}$$

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 $i = r - R_f$):

$$\mathsf{state}[0] = \mathsf{state}[0]^\alpha \oplus \mathrm{RoundConstants}'_r$$
$$\mathsf{state} = \mathsf{state} \times \mathcal{S}_{r - R_f}$$

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 $\mathsf{state}[1]$.

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

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

Algorithm Comments:

Line 1. Initialize the state.

Line 2. Adds the pre- $r = 0$ round constants.

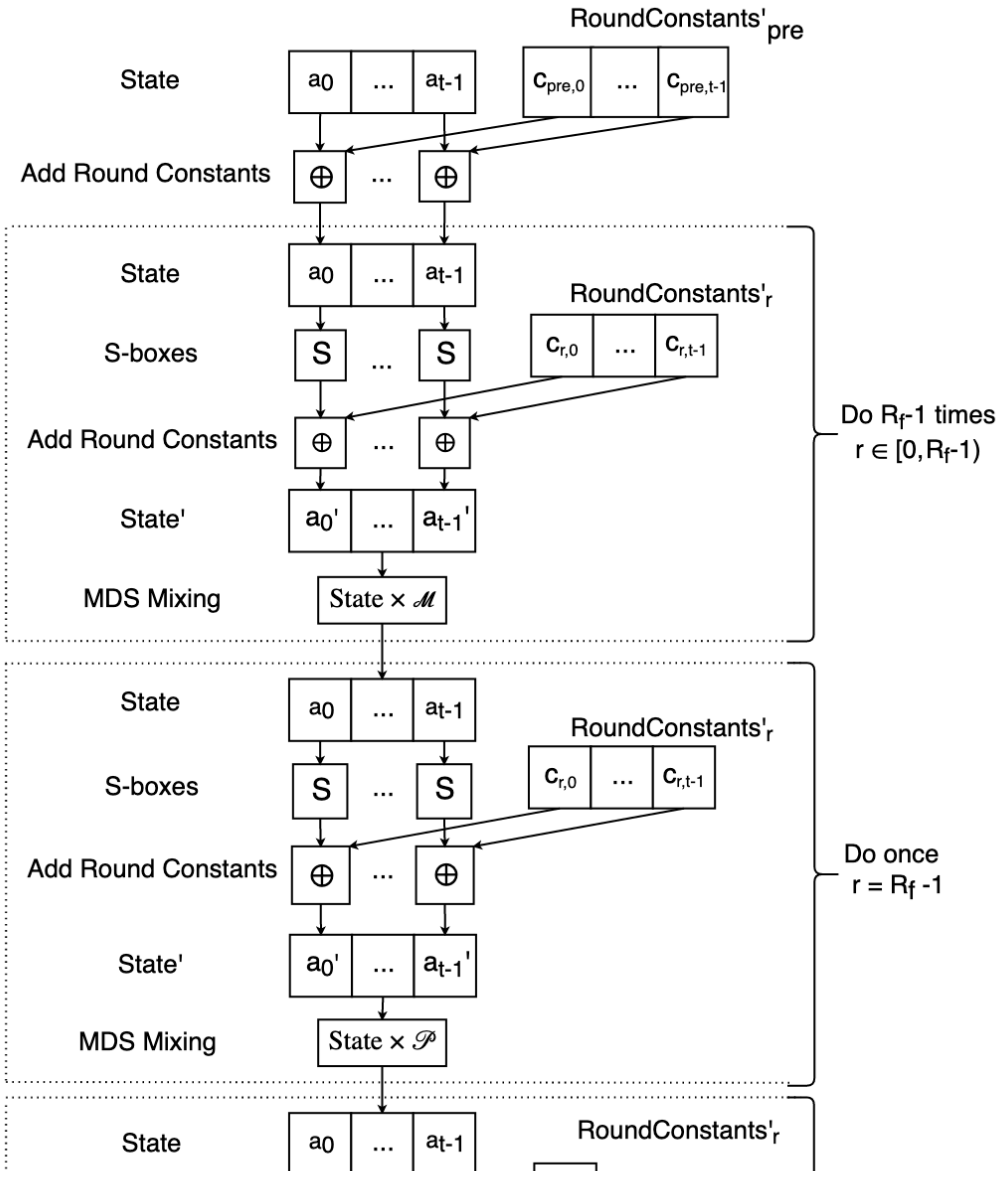
Lines 3-4. Performs all but the last first-half of full rounds $r \in [R_f - 1]$.

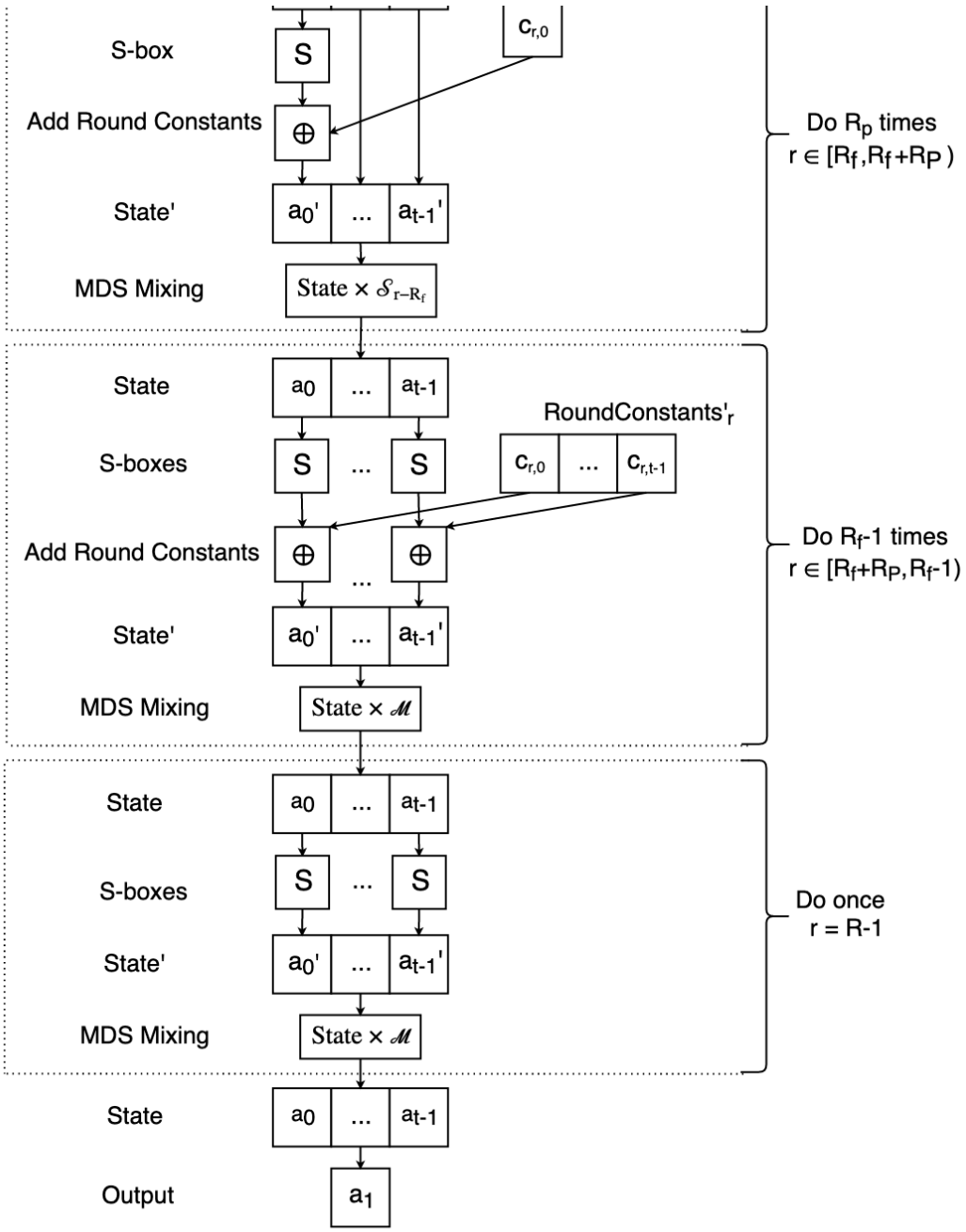
Line 5. Performs the last first-half full round $r = R_f - 1$.

Lines 6-8. Performs the partial rounds $r \in [R_f, R_f + R_P)$. Mixing in the i^{th} partial round, where $i = r - R_f$, is done using the i^{th} sparse matrix $\mathcal{S}_{r-R_f}$.

Lines 9-10. Performs all but the last second-half full rounds $r \in [R_f + R_P, R - 1)$.

Line 11. Performs the last second-half rull round $r = R - 1$.




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