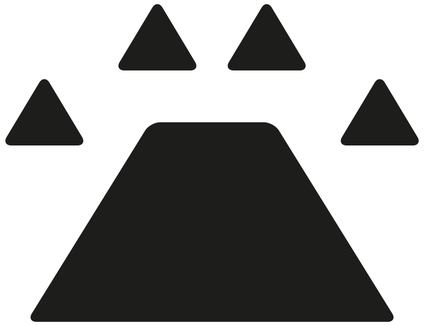
Foundations of High Speed Cryptography

Module 1 - Theory
Lesson 3 - Cryptographic Hashes and Merkle Trees



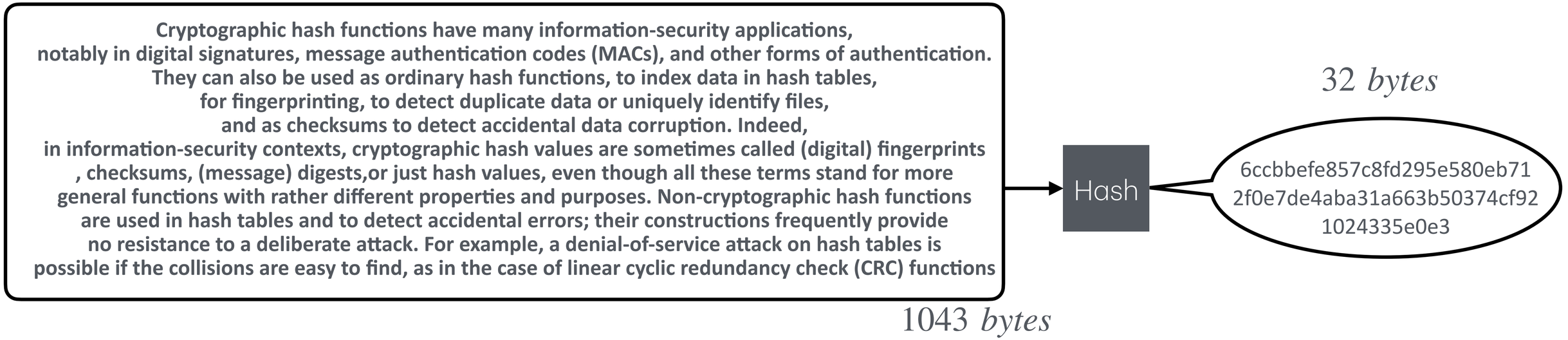
Karthik Inbasekar

Agenda

- **Motivation**
- **Cryptographic Hash Functions (CHF)**
  - Collision Resistant Hash Functions: Definitions and examples
  - Bitwise hashes: Sponge Construction, eg: SHA3
  - Finite Field friendly Hashes: General structure, eg: Poseidon2
- **Merkle Tree**
  - General principles
  - Merkle commitment Scheme
  - Optimizations
- **Hash Functions as Random Oracles**
  - Fiat-Shamir transformation

Motivation

- Cryptographic hash: **fundamental cryptographic primitive** that enables mapping any input into a fixed-size digest that “looks random”.



- **Useful building blocks** : Data integrity, Consensus: proof of work , Cryptographic commitment Schemes, Pseudo Random Number Generators, digital signatures, password verification

- Digests do not reveal actual contents of data, can be used for **validation of information while preserving privacy**.

Use Cases

- **Data integrity**: unique fingerprints of blocks/transactions in Ethereum

  Keccak : Account, Storage, Transaction and receipt Tries.

- **Proof of work:** Consensus mechanism in Bitcoin

  Double SHA-256 for hashing, allows decentralized agreement on transaction history.

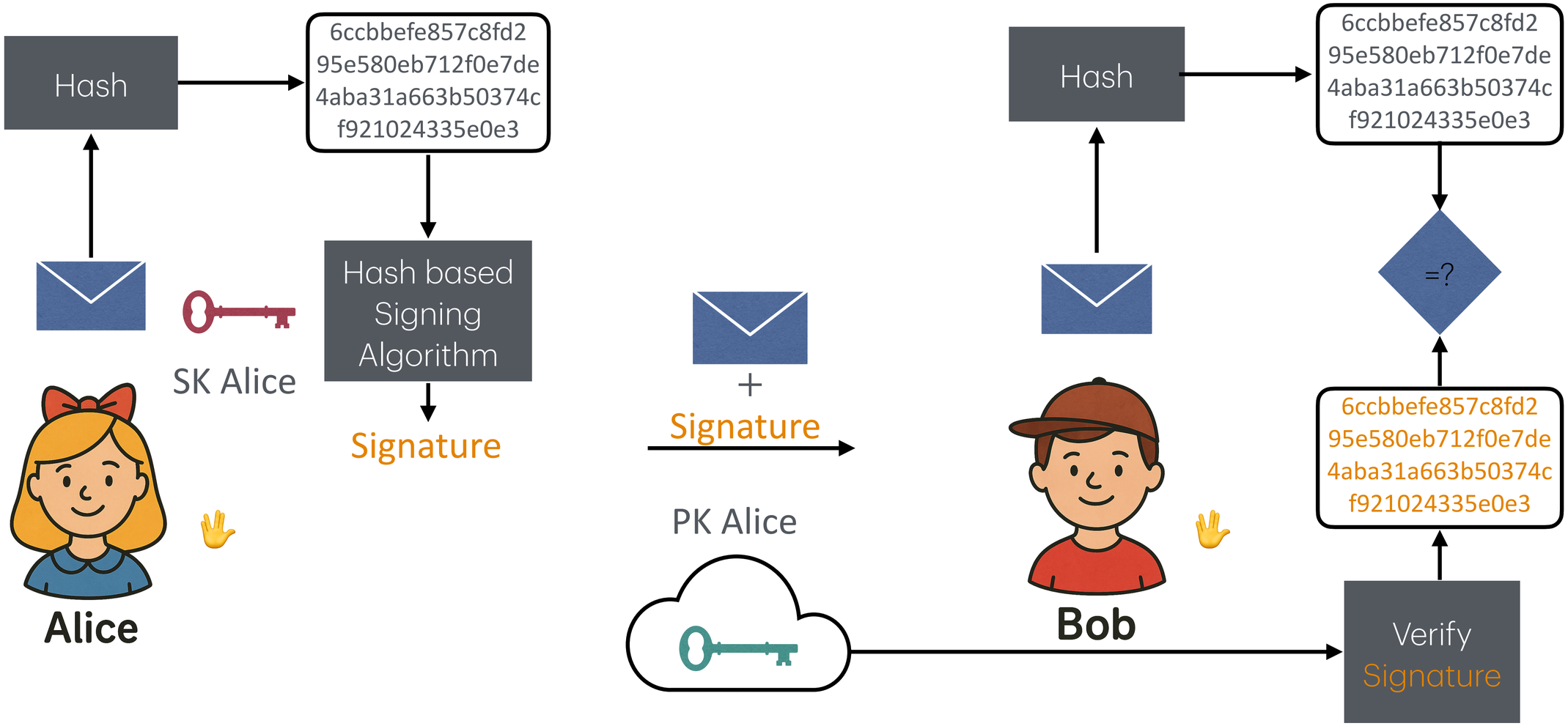
- **Cryptographic Commitment Schemes:** Hashes can act as binding and hiding commitments

  Merkle proofs : proving knowledge of data, used in Cryptographic proof systems.

- **Fiat Shamir Heuristic:** Turn interactive proof systems into non-interactive proof system.

- **Digital Signatures**: Post quantum digital signatures, are built on Cryptographic Hash Functions.
  Eg: XMSS, SPHINCS++

Digital Signatures using hash functions

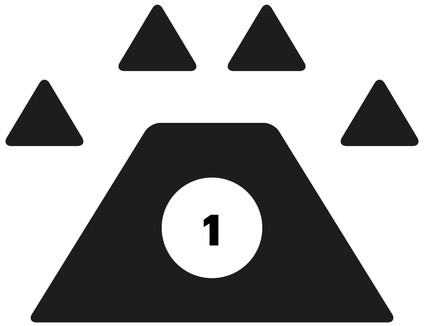


- Eg: XMSS , SPHINCS++

https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/papers/session5-hulsing-paper.pdf

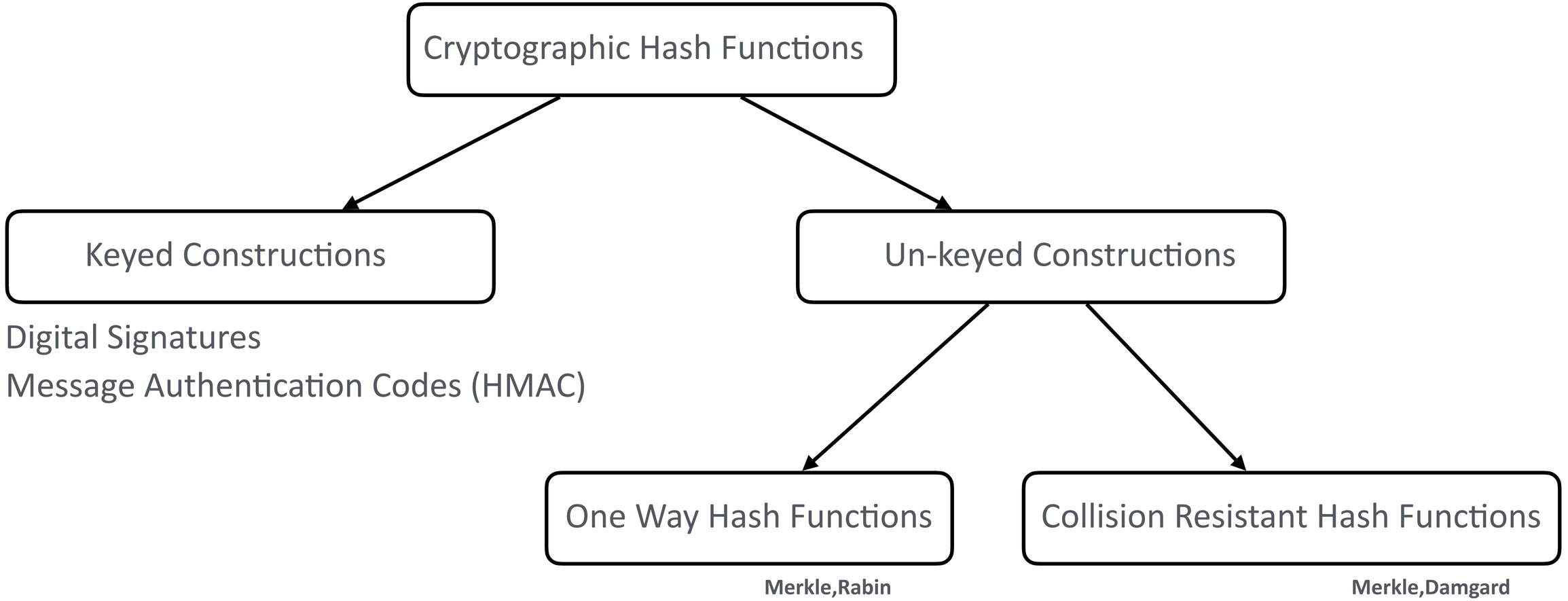
Foundations of High Speed Cryptography

Module 1 - Theory
Lesson 3 - Hashes and Merkle Trees



Cryptographic Hash functions

Taxonomy



Eg: Digital Signatures

Eg: Message Authentication Codes (HMAC)

Bert Preneel, CRYPTOGRAPHIC HASH FUNCTIONS, Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography, W. Wolfowicz (ed.), Fondazione Ugo Bordoni, pp. 161–171, 1993.

Collision Resistant Hash Functions

Given a hash h, the argument x can be of arbitrary length and the digest $y = h(x)$ of length $n \geq 256$

- **Pre-image resistance**

  Given y, h computationally infeasible to find $x \mid h(x) = y$

- **Second Pre-image resistance**

  Given $x, y = h(x)$ it is computationally infeasible to find $x' \mid h(x') = h(x)$

- **Collision resistance**

  It is computationally feasible to find any two distinct inputs $x \neq x' \mid h(x') = h(x)$

  Birthday attack $\sim 2^{n/2}$

- **Pseudo randomness**

  The output of the hash function must not have any detectable output patterns that leak information.

Preimage resistance vs Collision resistance

- Given $SHA256(x)$ is a known collision resistant function: what about this one?

$$h(x) = SHA256(x) \mod 2^{32}$$

- Given $g(x)$ is collision resistant, and with digest size $n-1$ bits

$$h(x) = \begin{cases} 1 \,||\, x \;,\; if\; len(x) = n-1 \\ 0 \,||\, g(x) \;,\; otherwise \end{cases}$$

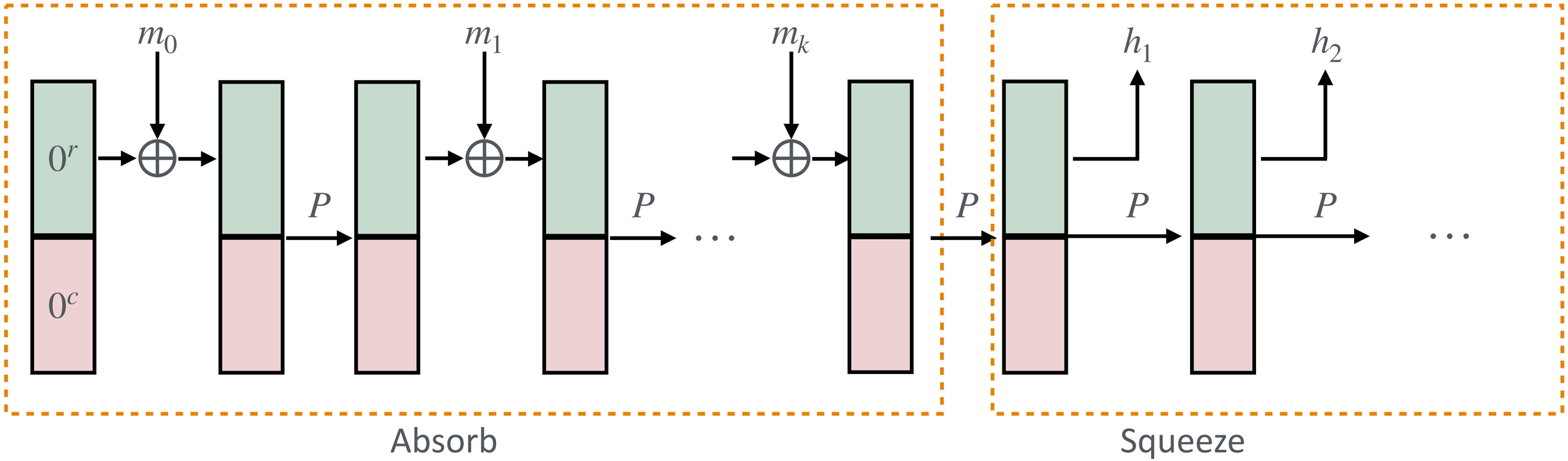
- Given $h(x) = x \forall\; x \in \{0,1\}^n$

- Given $g(x)$ is collision resistant (by some hard problem LWE/SIS/DLOG) such that

$$g(x_1) + g(x_2) = g(x_1 + x_2)$$ Eg SWIFFT

Bitwise hashes: SHA -3

- SHA3 uses the sponge construction to hash arbitrary length input to a fixed size output
- The hash state has a public part r: **rate** and a private part c: **capacity,** $r + c = 1600\ bits$
- The input is chunked into $m_0||m_1||m_2||\ldots||m_k||$ such that $|m_i| = r\ bits$



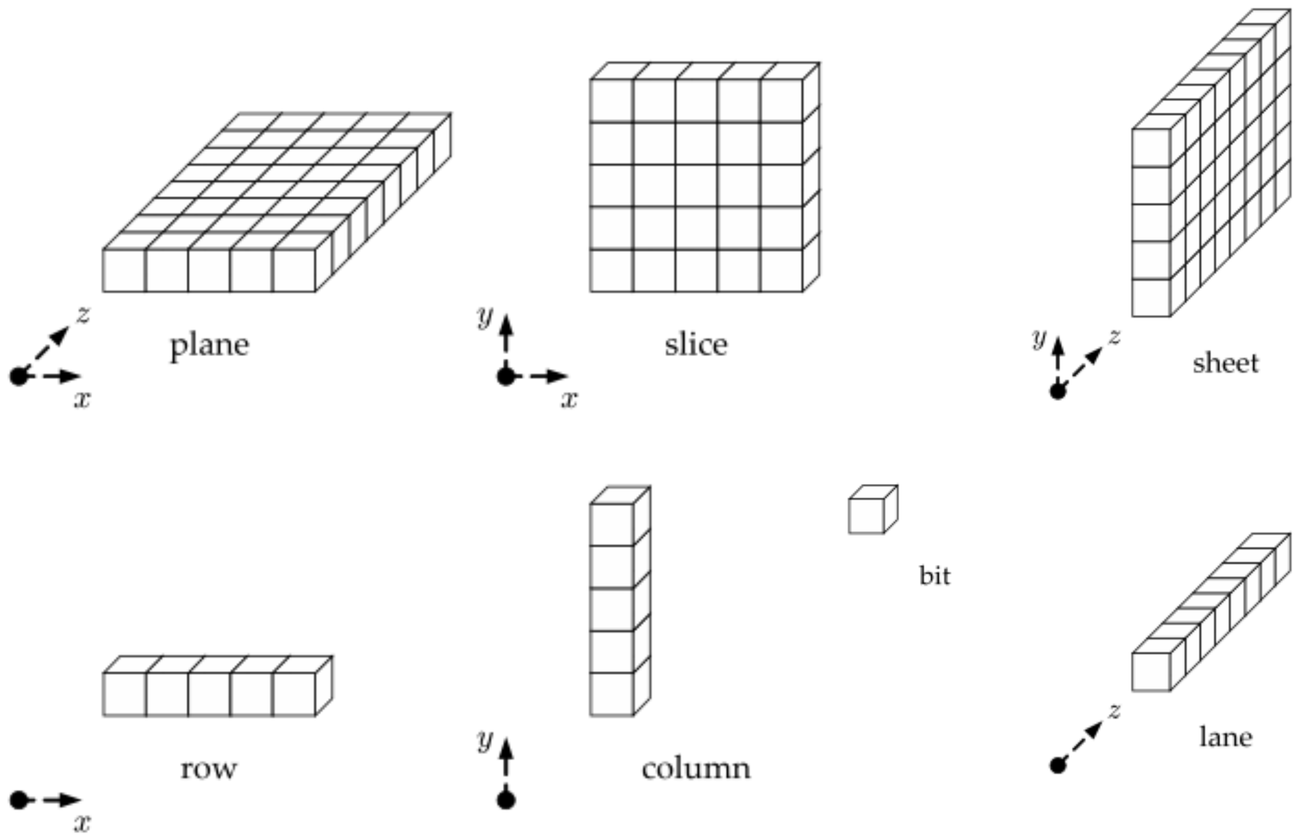
- The permute function P consists of multiple rounds, input chunks are XORed to rate, output chunks are read r bits at a time.

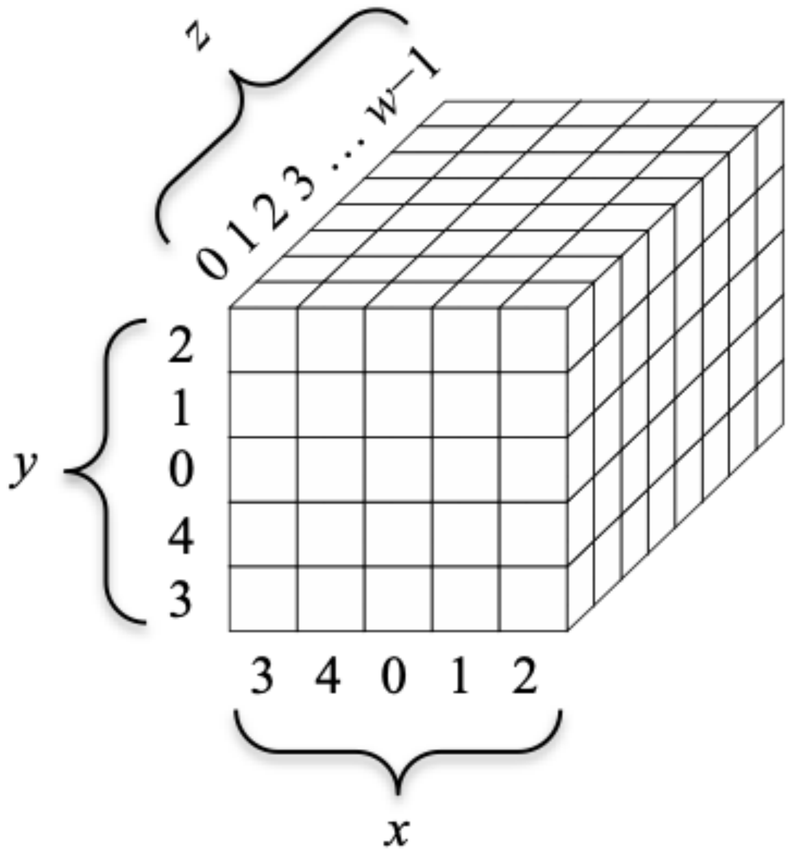
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

SHA-3 : Keccak-f permutation

- The hash state is represented as an array $5 \times 5 \times 64 = 1600$

$w = 2^l$, $l \in [0,6]$



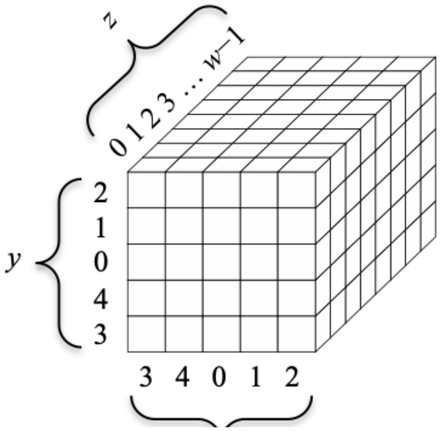


25 lanes each of 64 bit word size

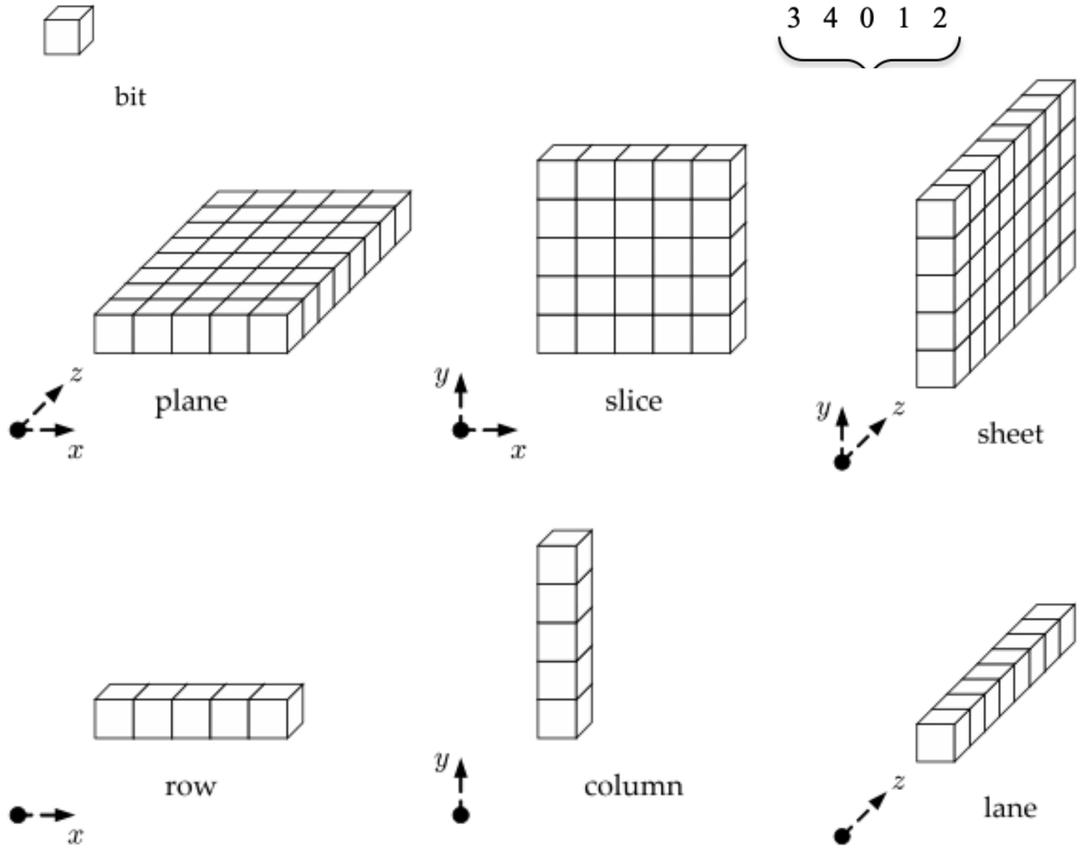
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

SHA-3 : Keccak-f permutation

- Each permute operation operates for $12 + 2l$ rounds in the five step sequence



Step	Operation	Optimization
Θ	For each bit, XOR with parity of neighboring columns	Parallelize : Column parity, XOR of bits with parity
P	Rotate each bit of a lane by an offset length	Parallelize: Bit rotation per lane
Π	Permute the positions of the Lanes	Pure reordering, parallel across all lanes
X	XOR each bit with a non linear function of two other bits in a row	Row wise parallel
I	Add round constants to the lane A[0,0,Z]	Store round constants in a cached read only memory



- Bit interleaving for area/speed tradeoffs: $25s$ words of $64/s$ bits for different hardware platforms.

Finite Field Hashes

- In cryptographic constructions involving Proofs of computational integrity, one is required to prove that a given hash state is valid, and the hash was executed correctly

- In such constructions, each bitwise operation of the hash is represented as an arithmetic circuit.

$$Q(A,B) = A + B - 2 \cdot A \cdot B$$

XOR Constraint equation

- For 64kb Sha256 we need to check about 30 M - 45 M constraint equations!!

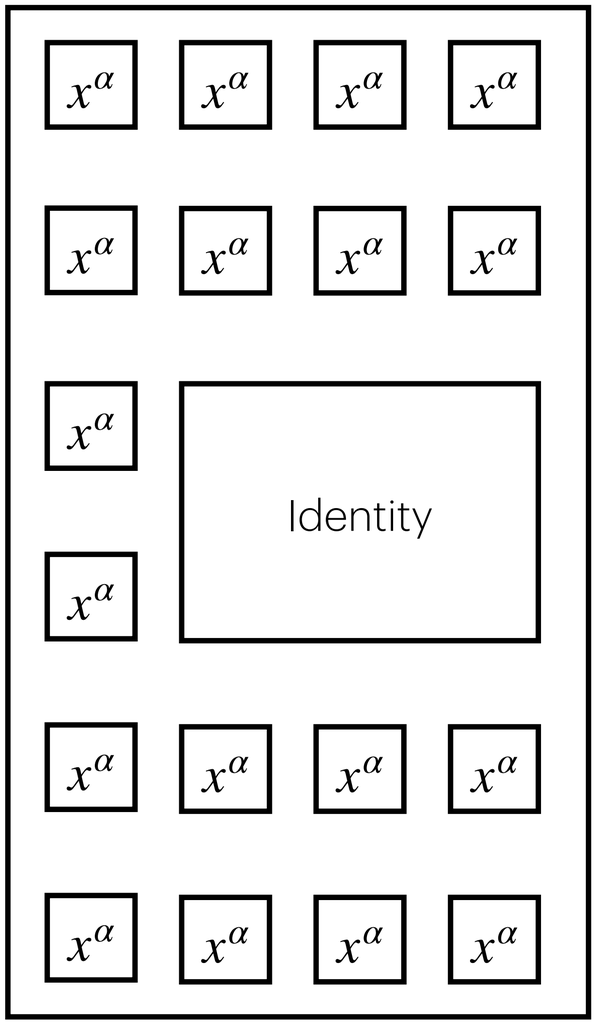
- Due to large number of bitwise operations, the arithmetic circuit representation consists of large number of gates, and provable constraints that increase prover overhead significantly,

- Finite Field hashes, operate on finite fields instead of bits, and have a much simpler arithmetic circuit representation.

https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/

Finite Field Hashes

- A popular construction is based on the HADES family of block ciphers



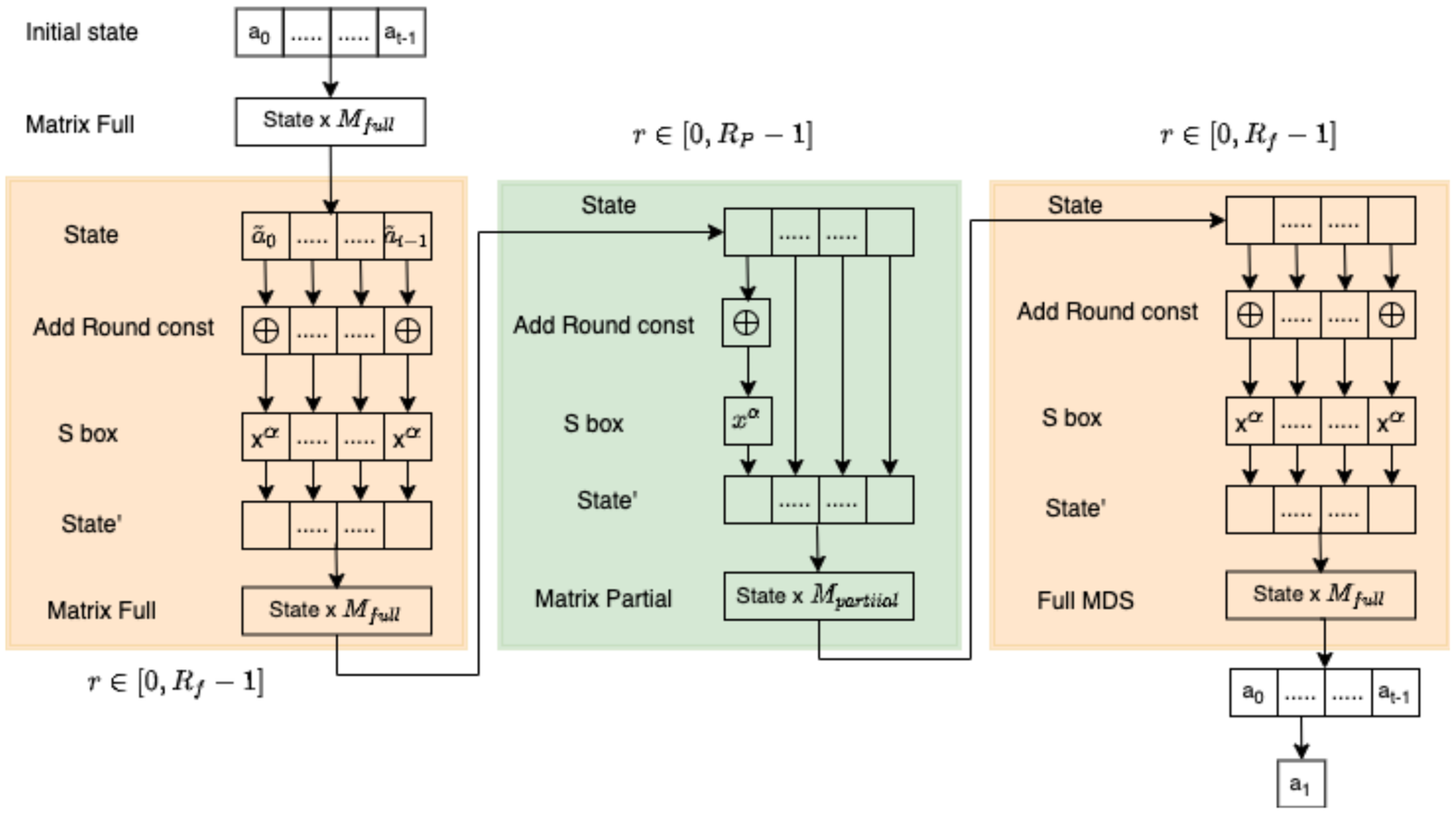
- The state is a vector of size t, with elements in $\mathbb{F}_p^t$
- Cipher operations
  - Linear layer: Add round constant vectors
  - Linear layer: Matrix multiplications by an MDS matrix
  - Full Non-Linear layers: $gcd(\alpha, p-1) = 1$

$$state[i] \rightarrow state[i]^\alpha \quad \forall i \in [0, t-1]$$

  - Partial Non-Linear layers:

$$state[0] \rightarrow state[0]^\alpha$$
$$state[i] \rightarrow state[i] \ \forall i \in (0, t-1]$$

Poseidon 2



Full round

Full round

Partial round

Digest

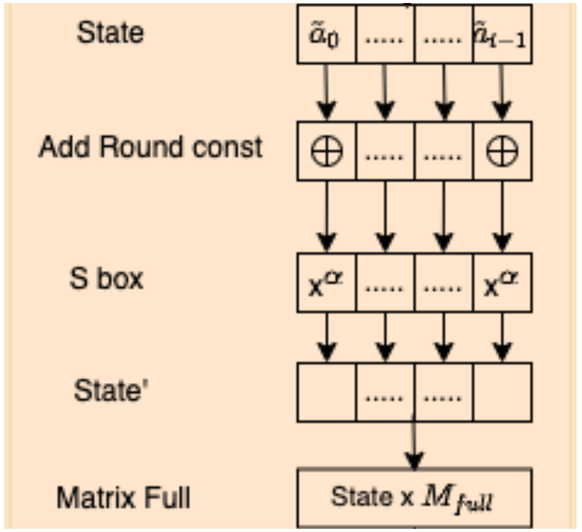
https://eprint.iacr.org/2023/323

https://github.com/ingonyama-zk/Poseidon_parameters/tree/master/Poseidon2

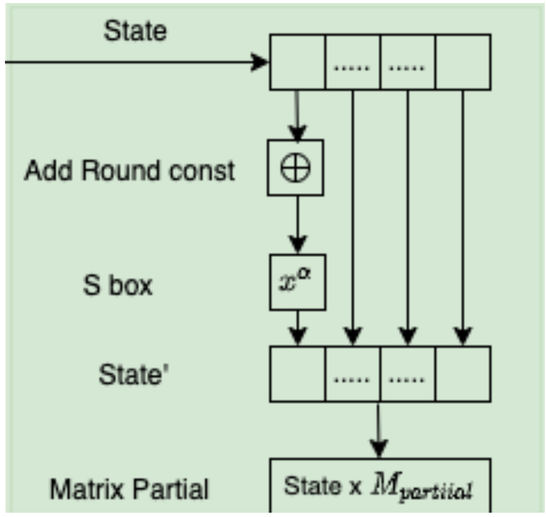
Poseidon 2

- The state is a vector of size $t \in \{2,3,4,\ldots,4 \cdot t', \ldots 24\}$, with elements in $\mathbb{F}_p^t$, $p > 2^{30}$

$$M_{full} = \begin{cases} M_4 & if\ t = 4 \\ circ(2 \cdot M_4, M_4, \ldots, M_4) \in \mathbb{F}_p^{t \times t} & if\ t \geq 8 \end{cases}$$



$$M_4 = \begin{pmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{pmatrix}$$



$$\mu_i \in \mathbb{F}_p \backslash \{0,1\}$$

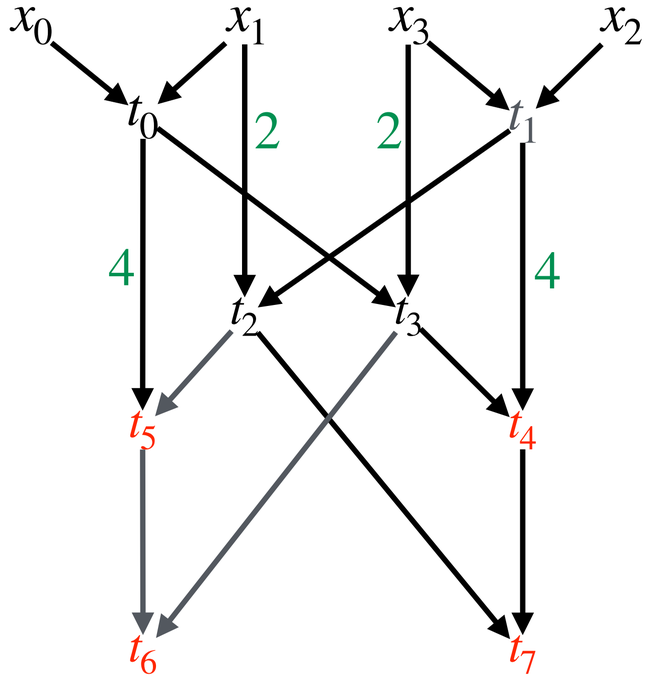
$$M_{partial} = \begin{pmatrix} \mu_0 & 1 & 1 & \ldots & 1 \\ 1 & \mu_1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & \ldots & \ldots & \mu_{t-1} \end{pmatrix}$$

https://eprint.iacr.org/2023/323

https://github.com/ingonyama-zk/Poseidon_parameters/tree/master/Poseidon2

Poseidon 2: Matrix optimizations

$$M_4 \cdot x = \begin{pmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} t_6 \\ t_5 \\ t_7 \\ t_4 \end{pmatrix}$$



- Can compute a block in 8 A + 4 M (bit shifts)

$$M_{partial} = \begin{pmatrix} \mu_0 & 1 & 1 & \dots & 1 \\ 1 & \mu_1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & \dots & \dots & \mu_{t-1} \end{pmatrix}$$

Notice: easy arithmetization!

- μ_i can also be chosen as powers of 2, to turn partial matrix mults to take advantage of bit shifts

Poseidon 2: Matrix optimizations

$$M_{full} = \begin{cases} M_4 & if\ t = 4 \\ circ(2 \cdot M_4, M_4, \ldots, M_4) \in \mathbb{F}_p^{t \times t} & if\ t \geq 8 \end{cases}$$

- Circulant matrix multiplications are efficiently performed using Fourier Transform operations.
- But in this case, it is block circulant, and each M_4 multiplication is highly optimized already.
- Using FFT's here is not worth it for the overhead they bring to the computation - not recommended for t sizes commonly used in production.
- However, the MDS matrices themselves are not sacred, and one can find many useful representations for special fields like Goldilocks and M31.

https://eprint.iacr.org/2023/323