The Tip5 Hash Function for Recursive STARKs

Alan Szepieniec
艾伦·佘丕涅茨
alan@neptune.cash
Neptune

Alexander Lemmens
Alexander.Lemmens@vub.be
DIMA, Vrije Universiteit Brussel

Jan Ferdinand Sauer
ferdinand@neptune.cash
Neptune

Bobbin Threadbare
bobbinth@protonmail.com
Polygon

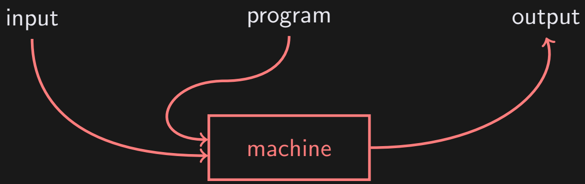
Al-Kindi
al-kindi-0@protonmail.com
Polygon

neptune

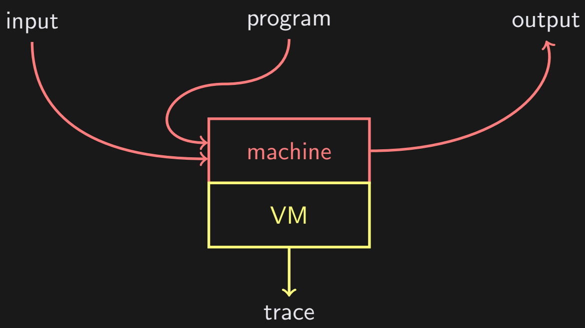
Triton VM

STARK

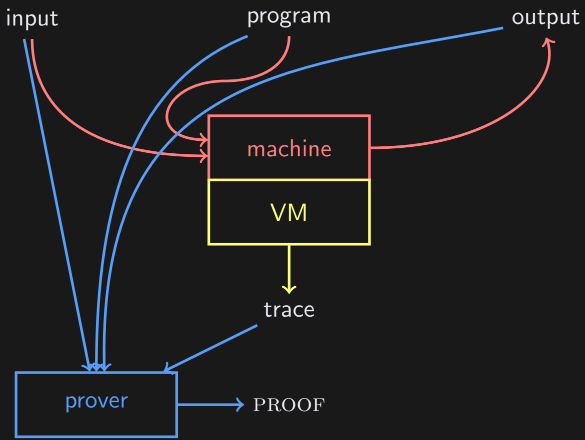
STARK



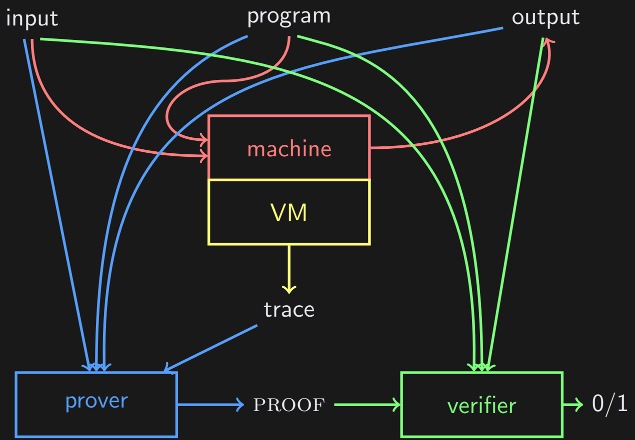
STARK



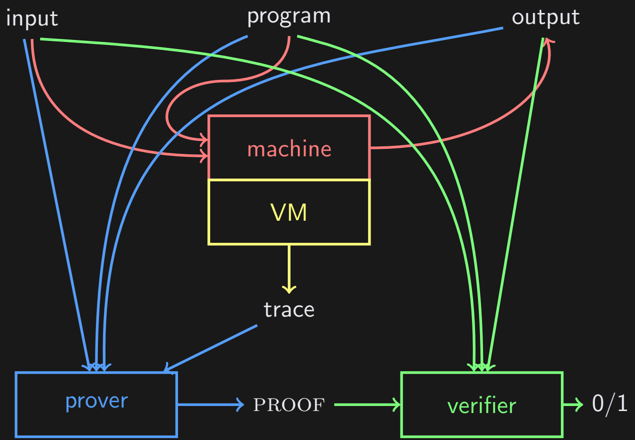
STARK



STARK


input
program
output
machine
VM
trace
prover
PROOF
verifier
0/1

STARK



✓ fast verifier
✓ transparent
✓ zero-knowledge
✓ post-quantum

Recursive STARK

input*:

(input, program, output, PROOF)

Recursive STARK

input*:

(input, program, output, PROOF)

program*:

verifier

Recursive STARK

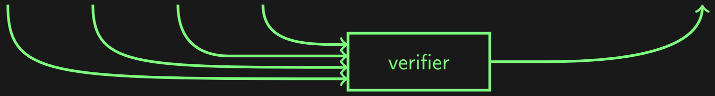
input*:	program*:	output*:
(input, program, output, PROOF)	verifier	0/1

Recursive STARK

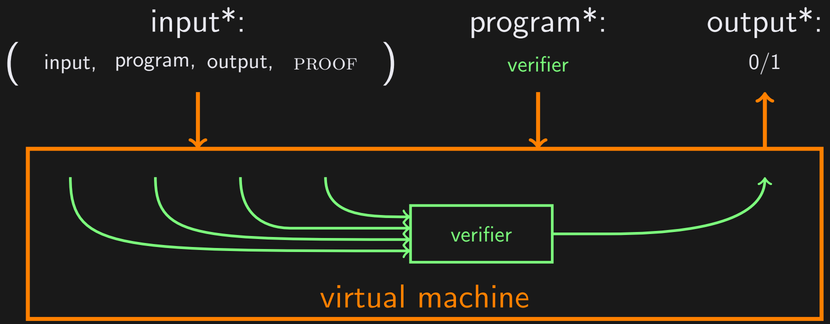
input*: (input, program, output, PROOF)

program*: verifier

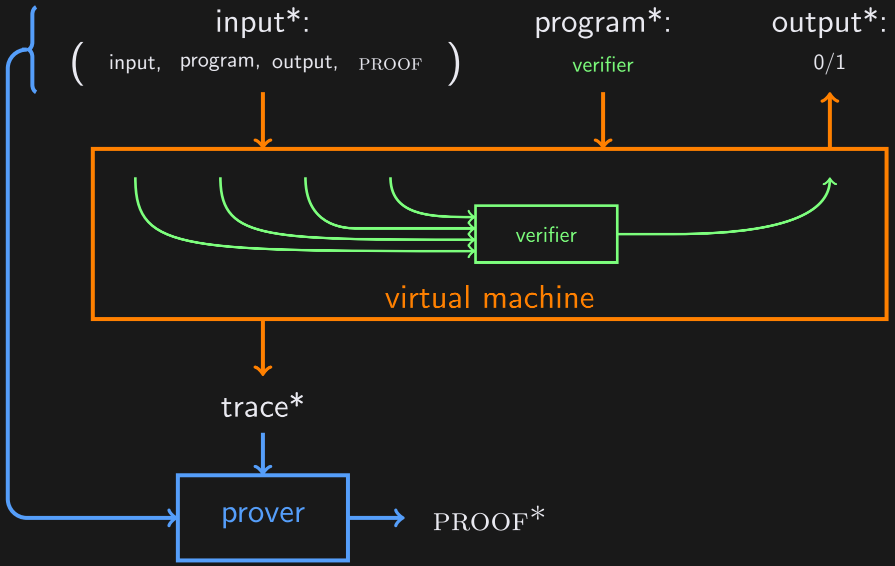
output*: 0/1



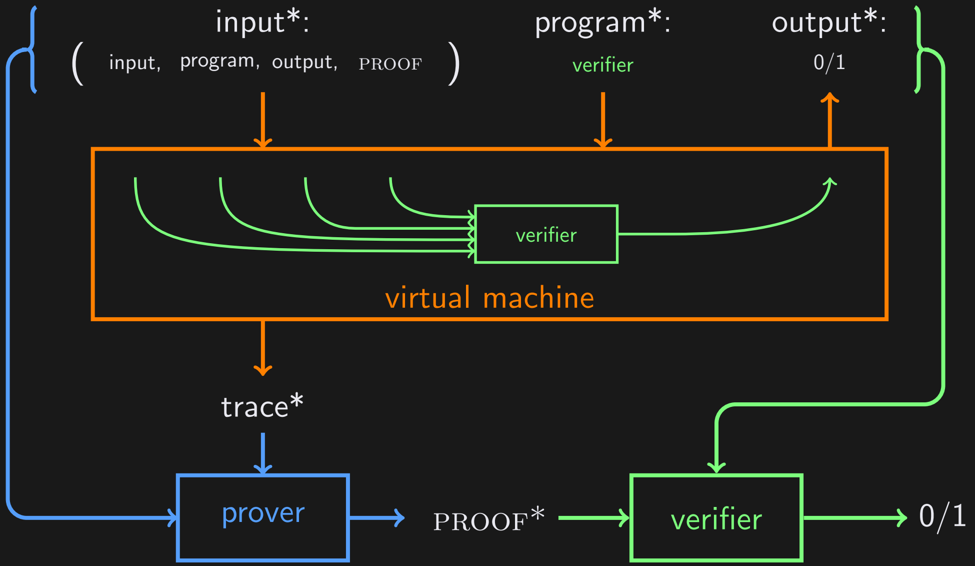
Recursive STARK



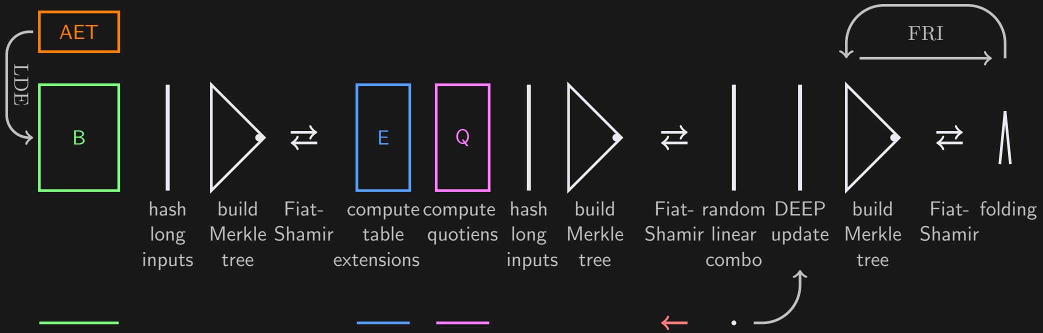
Recursive STARK



Recursive STARK



STARK Workflow



STARK Cost

Prover:

- LDE
- trace arithmetic
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- evaluate AIR
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- FRI

STARK Cost

Prover:

- **LDE**
- trace arithmetic
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- evaluate AIR
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- **FRI**

STARK Cost

Prover:

- **LDE**
- trace arithmetic
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- evaluate AIR
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- **FRI**

Theory predicts LDE is the bottleneck ...

... but 80% of the time is spent hashing.

STARK Cost

Prover:

- LDE
- trace arithmetic
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- evaluate AIR
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- FRI

So just use Blake3 ... right?

Theory predicts LDE is the bottleneck ...

... but 80% of the time is spent hashing.

STARK Cost

Prover:

- LDE
- trace arithmetic
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- evaluate AIR
- hashing long inputs
- building Merkle tree
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- FRI

So just use Blake3 ... right?

Verifier:

- Fiat-Shamir
- evaluate AIR
- DEEP update
- Fiat-Shamir
- verify Merkle path
- FRI colinearity check
- hashing long inputs

Theory predicts LDE is the bottleneck ...

... but 80% of the time is spent hashing.

STARK Cost

Prover:

- **LDE**
- trace arithmetic
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- evaluate AIR
- **hashing long inputs**
- **building Merkle tree**
- Fiat-Shamir
- out-of-domain evaluation
- evaluate AIR
- random linear combination
- DEEP update
- **FRI**

Theory predicts LDE is the bottleneck ...

... but 80% of the time is spent hashing.

So just use Blake3 ... right?

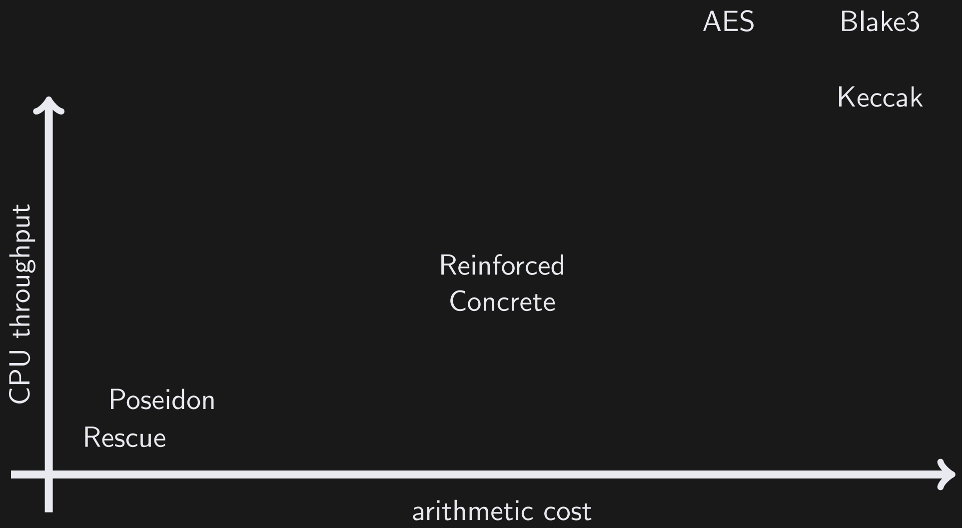
Verifier:

- Fiat-Shamir
- evaluate AIR
- DEEP update
- Fiat-Shamir
- verify Merkle path
- FRI colinearity check
- hashing long inputs

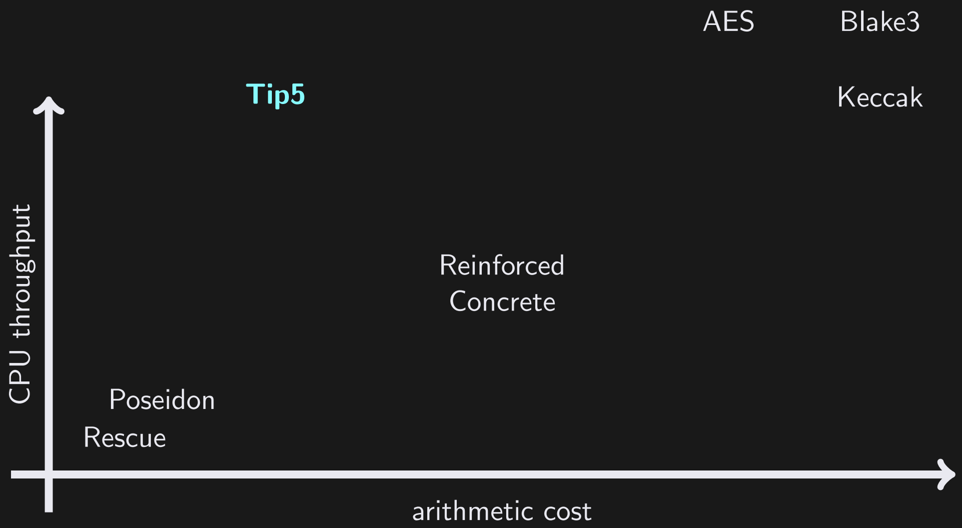
The VM must support hashing.

$\Rightarrow$ **we need an arithmetization-friendly hash function.**

Hash Function Orientation



Hash Function Orientation



Security of Arithmetization-Oriented Hash Functions

1. Statistical Cryptanalysis ✓✓✓
2. Algebraic Cryptanalysis ?????
   - in particular, *Gröbner basis* algorithms

Interlude: Gröbner Basis Algorithms

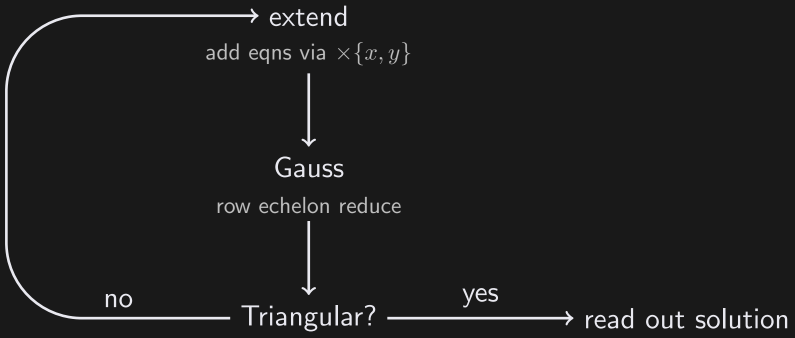
$$\left.\begin{aligned} x^3 - xy + 2y^2 - x + 1 = 0 \\ y^3 + 2x^2 + y^2 - x - y - 1 = 0 \end{aligned}\right\}$$

Interlude: Gröbner Basis Algorithms

$$\left.\begin{array}{r} x^3 - xy + 2y^2 - x + 1 = 0 \\ y^3 + 2x^2 + y^2 - x - y - 1 = 0 \end{array}\right\} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 & 1 & -1 & -1 & -1 \end{pmatrix}$$

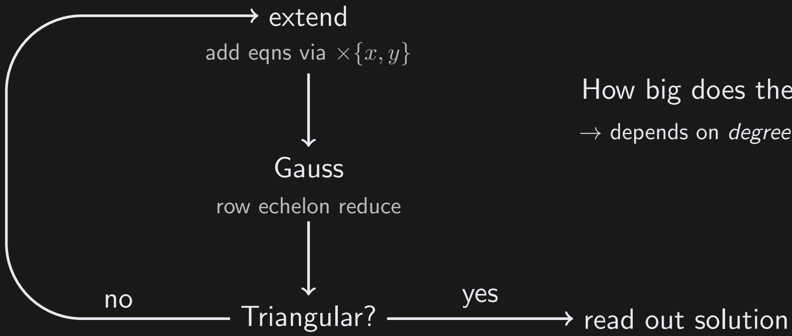
Interlude: Gröbner Basis Algorithms

$$\left.\begin{aligned} x^3 - xy + 2y^2 - x + 1 = 0 \\ y^3 + 2x^2 + y^2 - x - y - 1 = 0 \end{aligned}\right\} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 & 1 & -1 & -1 & -1 \end{pmatrix}$$



Interlude: Gröbner Basis Algorithms

$$\left.\begin{aligned} x^3 - xy + 2y^2 - x + 1 = 0 \\ y^3 + 2x^2 + y^2 - x - y - 1 = 0 \end{aligned}\right\} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 & 1 & -1 & -1 & -1 \end{pmatrix}$$



How big does the matrix get?

$\rightarrow$ depends on *degree of regularity*

Gröbner Basis Algorithms and Arithmetization-Oriented Hash Functions

1. Hash function is arithmetization-oriented
   $\Rightarrow$ it has a compact multivariate polynomial description
   $\Rightarrow$ Gröbner basis algorithms are relevant

Gröbner Basis Algorithms and Arithmetization-Oriented Hash Functions

1. Hash function is arithmetization-oriented

   $\Rightarrow$ it has a compact multivariate polynomial description

   $\Rightarrow$ Gröbner basis algorithms are relevant
2. Poseidon
   - Designers assume worst case d_{reg}
   - *this hypothesis has been falsified*

Gröbner Basis Algorithms and Arithmetization-Oriented Hash Functions

1. Hash function is arithmetization-oriented

   $\Rightarrow$ it has a compact multivariate polynomial description

   $\Rightarrow$ Gröbner basis algorithms are relevant
2. Poseidon
   - Designers assume worst case d_{reg}
   - *this hypothesis has been falsified*
3. Rescue
   - Designers extrapolate d_{reg} from small instances
   - *extrapolation hypothesis very questionable*

Gröbner Basis Algorithms and Arithmetization-Oriented Hash Functions

1. Hash function is arithmetization-oriented

   $\Rightarrow$ it has a compact multivariate polynomial description

   $\Rightarrow$ Gröbner basis algorithms are relevant
2. Poseidon
   - Designers assume worst case d_{reg}
   - *this hypothesis has been falsified*
3. Rescue
   - Designers extrapolate d_{reg} from small instances
   - *extrapolation hypothesis very questionable*
4. Reinforced Concrete
   - **lookup gates seem to defy GB attacks**

Design Criteria

Reinforced Concrete:

We Need:

Design Criteria

Reinforced Concrete:

- $p \in \{p_{\mathrm{BLS381}}, p_{\mathrm{BN254}}, p_{\mathrm{ST}}\}$

We Need:

- $p = 2^{64} - 2^{32} + 1$

Design Criteria

Reinforced Concrete:

- $p \in \{p_{\text{BLS381}}, p_{\text{BN254}}, p_{\text{ST}}\}$
- non-uniform rounds:
  $(C \circ B)^n \circ C \circ \text{Bars} \circ C \circ (B \circ C)^n$

We Need:

- $p = 2^{64} - 2^{32} + 1$
- uniform rounds: R^n

Design Criteria

Reinforced Concrete:

- $p \in \{p_{\mathrm{BLS381}}, p_{\mathrm{BN254}}, p_{\mathrm{ST}}\}$
- non-uniform rounds: $(C \circ B)^n \circ C \circ \mathrm{Bars} \circ C \circ (B \circ C)^n$
- many rounds + narrow state

We Need:

- $p = 2^{64} - 2^{32} + 1$
- uniform rounds: R^n
- few rounds + wide state

Design Criteria

Reinforced Concrete:

- $p \in \{p_{\mathrm{BLS381}}, p_{\mathrm{BN254}}, p_{\mathrm{ST}}\}$
- non-uniform rounds:
  $(C \circ B)^n \circ C \circ \mathrm{Bars} \circ C \circ (B \circ C)^n$
- many rounds + narrow state

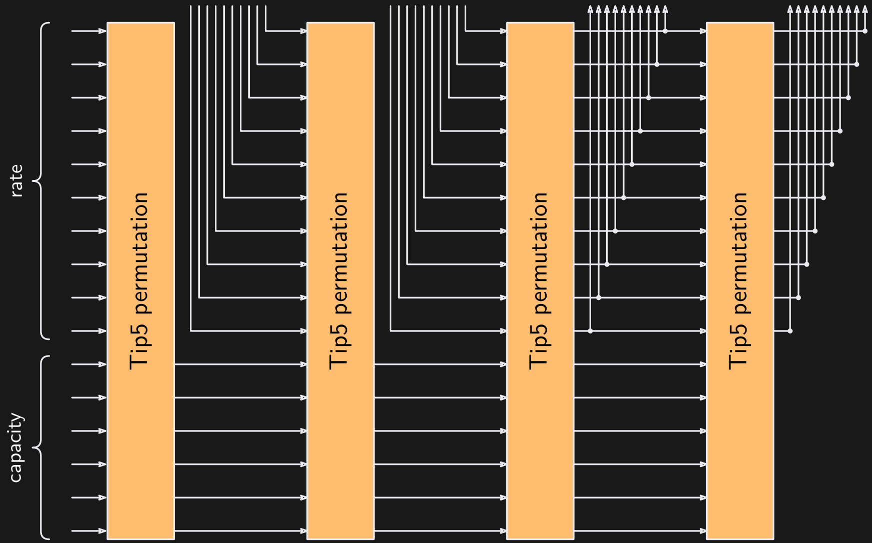
We Need:

- $p = 2^{64} - 2^{32} + 1$
- uniform rounds: R^n
- few rounds + wide state

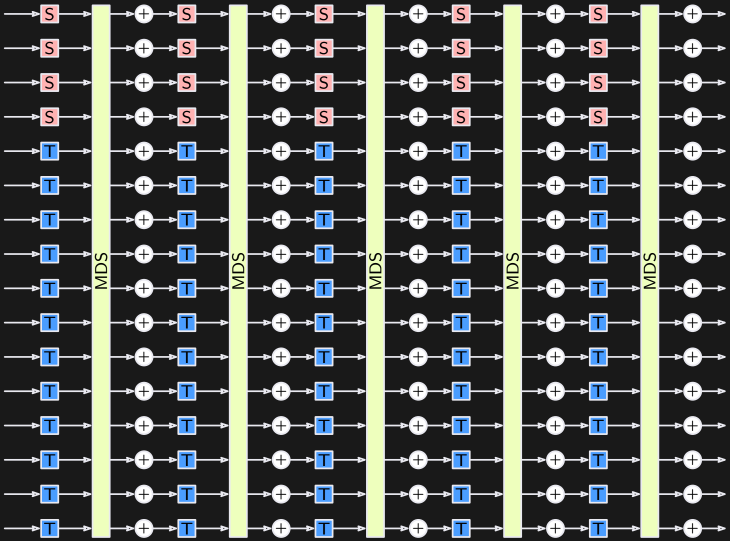
↑

Tip5 design constraints

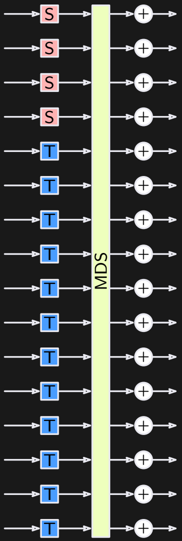
Tip5 Sponge Construction



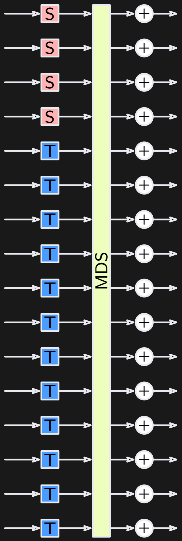
Tip5 – Permutation



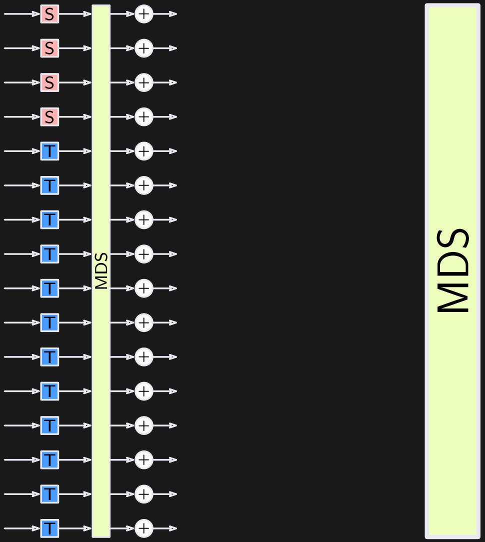
Tip5 – Permutation



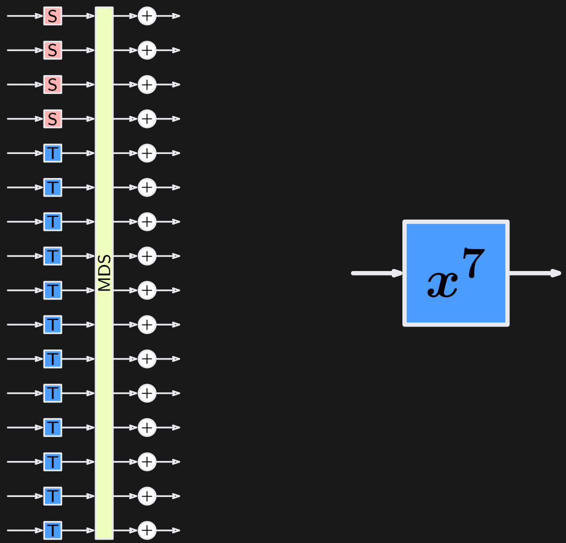
Tip5 – Permutation



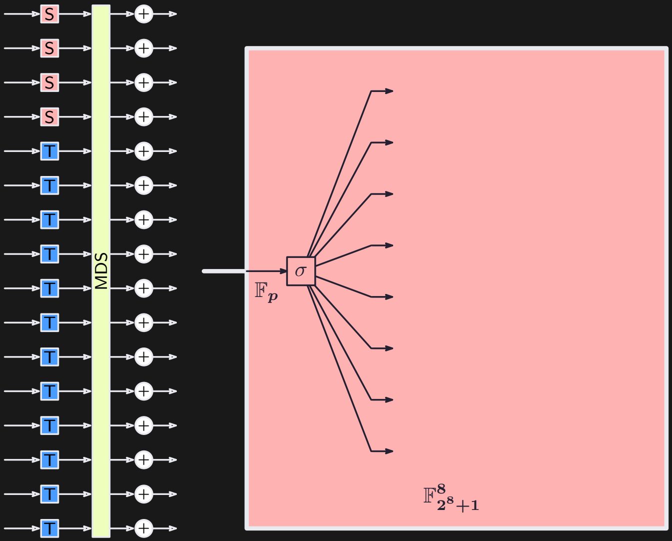
Tip5 – Permutation



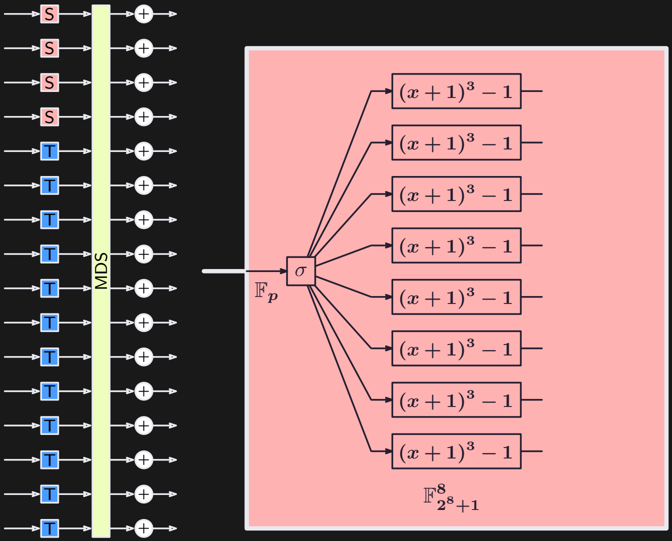
Tip5 – Permutation



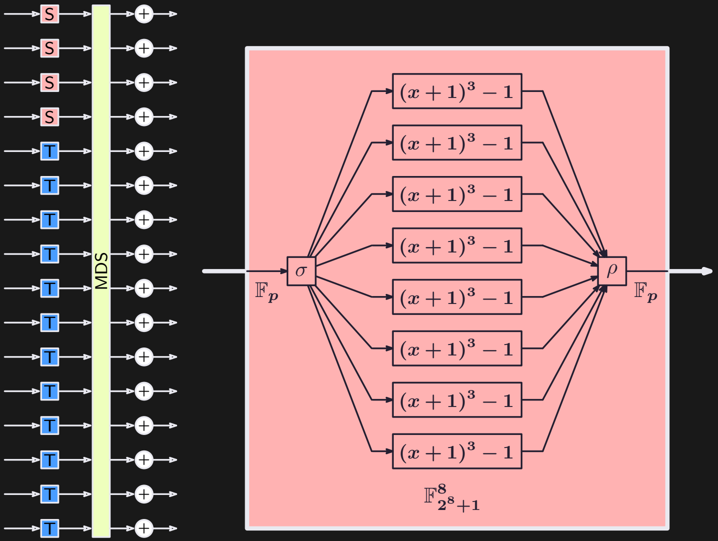
Tip5 – Permutation



Tip5 – Permutation

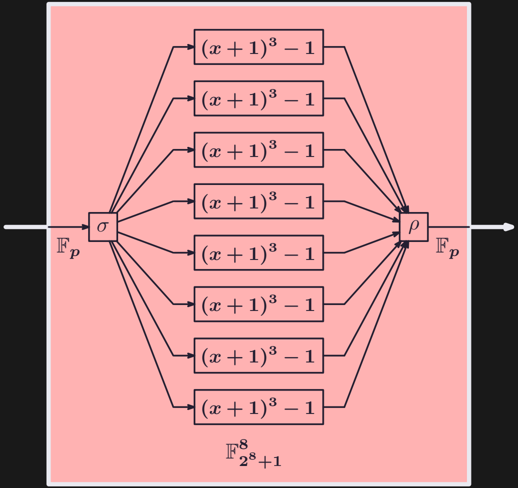


Tip5 – Permutation



How is this a permutation?

How is this a permutation?



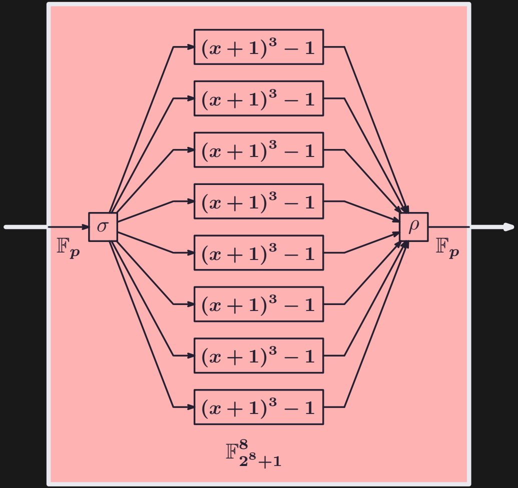
$L : \mathbb{F}_{257} \to \mathbb{F}_{257}\,,\;\; x \mapsto (x+1)^3 - 1$

– is a permutation on F_{257}

– has fixed points $\{0, 255, 256\}$

$\Rightarrow$ also a permutation on $\{0, \ldots, 255\}$

How is this a permutation?



$L : \mathbb{F}_{257} \to \mathbb{F}_{257}, \; x \mapsto (x+1)^3 - 1$

- is a permutation on F_{257}
- has fixed points $\{0, 255, 256\}$

$\Rightarrow$ also a permutation on $\{0, \ldots, 255\}$

Consider $S = \sigma \circ L^8 \circ \rho$ as a map on $\mathbb{N}_{<2^{64}}$

- $x = p - 1 \Leftrightarrow x =$ `0xffffffff00000000` maps to `0xffffffff00000000` $= p - 1$
- $x \geqslant p \Leftrightarrow x =$ `0xffffffff********` maps to `0xffffffff********` $\geqslant p$

$\Rightarrow x < p$ maps to $S(x) < p$ $\square$

Circulant MDS Matrix (1)

$$\begin{pmatrix} a & b & c \\ c & a & b \\ b & c & a \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} \quad \leftrightarrow \quad (a + bX + cX^2) \times (i + jX + kX^2) \mod X^3 - 1$$

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

$\rightarrow$ avoids modular reduction

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

→ avoids modular reduction

state vector first column

$$f \quad \times \quad g \quad \mod X^{16} - 1$$

$$\hookrightarrow f \times g \mod X^8 + 1$$

$$\hookrightarrow f \times g \mod X^8 - 1$$

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

$\rightarrow$ avoids modular reduction

state vector first column

$$f \quad \times \quad g \qquad \bmod X^{16} - 1$$

$\hookrightarrow$ $f \times g \mod X^8 + 1 \longrightarrow$ Karatsuba

$\hookrightarrow$ $f \times g \mod X^8 - 1$

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

$\rightarrow$ avoids modular reduction

state vector first column

$$f \times g \mod X^{16} - 1$$

- $f \times g \mod X^8 + 1 \longrightarrow$ Karatsuba
- $f \times g \mod X^8 - 1$
  - $f \times g \mod X^4 + 1$
  - $f \times g \mod X^4 - 1$

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

$\rightarrow$ avoids modular reduction

state vector first column

$$f \times g \mod X^{16} - 1$$

- $f \times g \mod X^8 + 1 \longrightarrow$ Karatsuba
- $f \times g \mod X^8 - 1$
  - $f \times g \mod X^4 + 1 \longrightarrow$ Karatsuba
  - $f \times g \mod X^4 - 1$

Circulant MDS Matrix (2)

compute matrix-vector product *over the integers*

$\rightarrow$ avoids modular reduction

state vector   first column

$$f \quad \times \quad g \qquad \bmod X^{16}-1$$

- $\longrightarrow f \times g \bmod X^8+1 \quad \longrightarrow \quad$ Karatsuba
- $\longrightarrow f \times g \bmod X^8-1$
  - $\longrightarrow f \times g \bmod X^4+1 \quad \longrightarrow \quad$ Karatsuba
  - $\longrightarrow f \times g \bmod X^4-1$
    - $\longrightarrow \ldots$
    - $\longrightarrow \ldots$

LogUp

Client

input	output
1	a
2	b
3	c
1	a

Server

input	output
1	a
2	b
3	c
4	d

LogUp

Client

input	output	comboc
1	a	$1 + a\alpha$
2	b	$2 + b\alpha$
3	c	$3 + c\alpha$
1	a	$1 + a\alpha$

Server

input	output	combos
1	a	$1 + a\alpha$
2	b	$2 + b\alpha$
3	c	$3 + c\alpha$
4	d	$1 + a\alpha$

$$\texttt{combo} = \texttt{input} + \alpha \cdot \texttt{output}$$

LogUp

Client

input	output	comboc
1	a	$1 + a\alpha$
2	b	$2 + b\alpha$
3	c	$3 + c\alpha$
1	a	$1 + a\alpha$

Server

input	output	combos	mult.
1	a	$1 + a\alpha$	2
2	b	$2 + b\alpha$	1
3	c	$3 + c\alpha$	1
4	d	$1 + a\alpha$	0

$$\texttt{combo} = \texttt{input} + \alpha \cdot \texttt{output}$$

LogUp

Client

input	output	comboc	ldc
1	a	$1+a\alpha$	$-$
2	b	$2+b\alpha$	$-$
3	c	$3+c\alpha$	$-$
1	a	$1+a\alpha$	$-$

Server

input	output	combos	mult.
1	a	$1+a\alpha$	2
2	b	$2+b\alpha$	1
3	c	$3+c\alpha$	1
4	d	$1+a\alpha$	0

$$\texttt{combo} = \texttt{input} + \alpha\cdot\ \texttt{output}$$

$$\texttt{ldc}_i = \texttt{ldc}_{i-1} + \frac{1}{\beta - \texttt{comboc}_i} \text{ and } \texttt{ldc}_0 = \frac{1}{\beta - \texttt{comboc}_0}$$

LogUp

Client

input	output	comboc	ldc
1	a	$1 + a\alpha$	–
2	b	$2 + b\alpha$	–
3	c	$3 + c\alpha$	–
1	a	$1 + a\alpha$	–

Server

input	output	combos	mult.	sum
1	a	$1 + a\alpha$	2	–
2	b	$2 + b\alpha$	1	–
3	c	$3 + c\alpha$	1	–
4	d	$1 + a\alpha$	0	–

$$\texttt{combo} = \texttt{input} + \alpha \cdot \texttt{output}$$

$$\texttt{ldc}_i = \texttt{ldc}_{i-1} + \frac{1}{\beta - \texttt{comboc}_i} \text{ and } \texttt{ldc}_0 = \frac{1}{\beta - \texttt{comboc}_0}$$

$$\texttt{sum}_i = \texttt{sum}_{i-1} + \frac{\texttt{mult}_i}{\beta - \texttt{combos}_i} \text{ and } \texttt{sum}_0 = \frac{\texttt{mult}_0}{\beta - \texttt{combos}_0}$$

LogUp

Client

input	output	comboc	ldc
1	a	$1 + a\alpha$	–
2	b	$2 + b\alpha$	–
3	c	$3 + c\alpha$	–
1	a	$1 + a\alpha$	–

Server

input	output	combos	mult.	sum
1	a	$1 + a\alpha$	2	–
2	b	$2 + b\alpha$	1	–
3	c	$3 + c\alpha$	1	–
4	d	$1 + a\alpha$	0	–

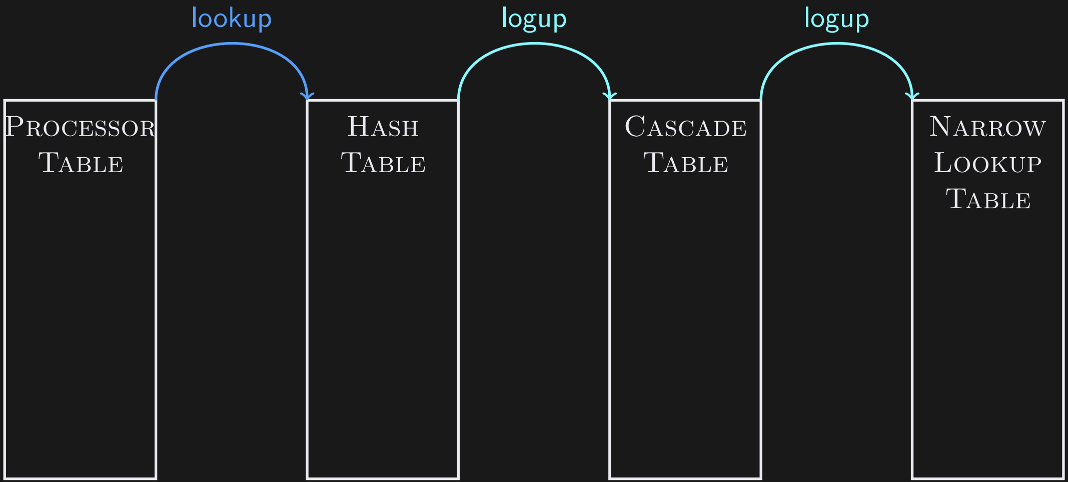
$$\texttt{combo} = \texttt{input} + \alpha \cdot \texttt{output}$$

$$\texttt{ldc}_i = \texttt{ldc}_{i-1} + \frac{1}{\beta - \texttt{comboc}_i} \text{ and } \texttt{ldc}_0 = \frac{1}{\beta - \texttt{comboc}_0}$$

$$\texttt{sum}_i = \texttt{sum}_{i-1} + \frac{\texttt{mult}_i}{\beta - \texttt{combos}_i} \text{ and } \texttt{sum}_0 = \frac{\texttt{mult}_0}{\beta - \texttt{combos}_0}$$

$$\sum_i \frac{1}{\beta - \texttt{comboc}_i} = \sum_i \frac{\texttt{mult}_i}{\beta - \texttt{combos}_i}$$

Cascade


lookup
logup
logup
Processor Table
Hash Table
Cascade Table
Narrow Lookup Table

Conclusion

Hash Function	Time [μs]
Rescue-Prime	18.186
Rescue-Prime Optimized	14.357
Poseidon	6.825
Tip5	0.850

Conclusion

Hash Function	Time [μs]
Rescue-Prime	18.186
Rescue-Prime Optimized	14.357
Poseidon	6.825
Tip5	0.850

$2.68\times$ speedup in **Triton VM**.

The Tip5 Hash Function for Recursive STARKs

Alan Szepieniec
艾伦·佘丕涅茨
alan@neptune.cash
Neptune

Alexander Lemmens
Alexander.Lemmens@vub.be
DIMA, Vrije Universiteit Brussel

Jan Ferdinand Sauer
ferdinand@neptune.cash
Neptune

Bobbin Threadbare
bobbinth@protonmail.com
Polygon

Al-Kindi
al-kindi-0@protonmail.com
Polygon

neptune

Triton VM