Monolith:
May the Speed of SHA-3 be With You

Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger,
Christian Rechberger, Markus Schofnegger, **Roman Walch**

20.09.2023

WWW.TACEO.IO

Motivation

- Cryptographic hash functions integral for ZK use cases
  - Membership in Merkle trees
    - Anonymous transactions/credentials
  - ZK rollups
  - Recursive SNARKs
- New hash functions optimized for ZK:
  - Poseidon, Rescue, Griffin, ...
- Performance:
  - ✔ Significantly outperform e.g. SHA-3 inside ZK circuits
  - ✖ Orders of magnitude slower for hashing

Plain Hashing Performance

- Plain hashing performance can be bottleneck
- Example: Recursive proofs (e.g., for signature aggregation using STARKs)
  - Build Merkle trees outside circuit and open them inside ZK circuits

  $\Rightarrow$ Slow hashing performance:

  Merkle tree generation up to half of overall proof time [COS20; Pol22]
- Hash functions design criteria:
  - Low number of multiplications for ZK performance
  - Low number of total operations for plain performance
  - How to achieve?

$\Rightarrow$ Use lookup arguments in modern proof systems

Plain Hashing Performance

- Plain hashing performance can be bottleneck
- Example: Recursive proofs (e.g., for signature aggregation using STARKs)
  - Build Merkle trees outside circuit and open them inside ZK circuits

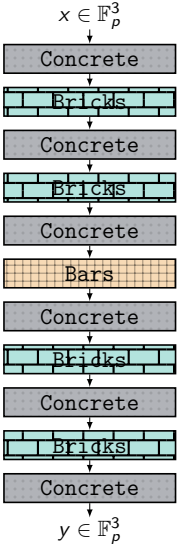
  $\Rightarrow$ Slow hashing performance:

  Merkle tree generation up to half of overall proof time [COS20; Pol22]
- Hash functions design criteria:
  - Low number of multiplications for ZK performance
  - Low number of total operations for plain performance
  - How to achieve?

$\Rightarrow$ Use lookup arguments in modern proof systems

Reinforced Concrete

- First arithmetization friendly hash function optimized for lookup tables
- 3 types of layer:
  - Concrete: Affine mixing
  - Bricks: Arithmetic non-linear layer
  - Bars: Decomposition and lookup table
    - Lookup represents repeated $(x + a)^d$
- Security:
  - One Bars layer for algebraic security
  - 6 Concrete / Bricks for statistical security



Reinforced Concrete (cont.)

- Faster than any previously published arithmetization oriented hash function
  - When using lookup tables
- But still significantly slower than e.g. SHA-3
- Fixed state size of 3 field elements
  - $\Rightarrow$ Only for large prime fields ≈ 256 bits
  - E.g., KZG based proof systems
- Decomposition is slow and difficult to generalize
- Arithmetic function in lookup table
  - Slow without lookup table (side channel attacks?)
  - Only efficient in proof systems with lookup tables

Goal

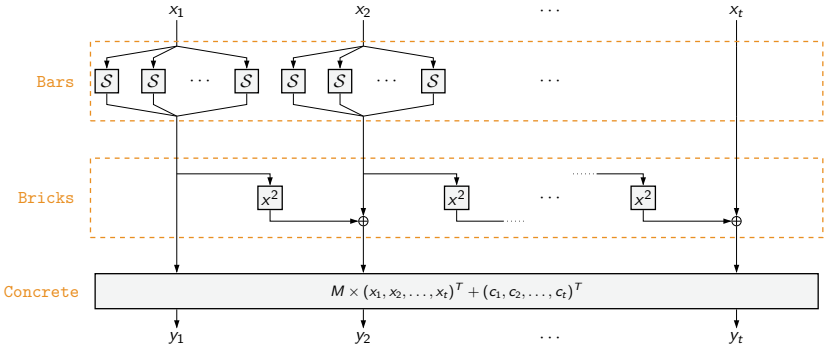
- New Hash function:
  - Efficient plain performance
    - Implementable without lookup tables to resist side-channel attacks
  - Efficient proof system representation
    - Utilize lookup arguments for fewer constraints
- Focus on FRI-based proof systems
  - Small prime-fields with fast modular reductions
    - $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$ (e.g. Plonky2 [Pol22])
    - $p_{\texttt{Mersenne}} = 2^{31} - 1$ (e.g. Plonky3 [Pol23])

$\Rightarrow$ Monolith

Monolith

The Monolith Permutation

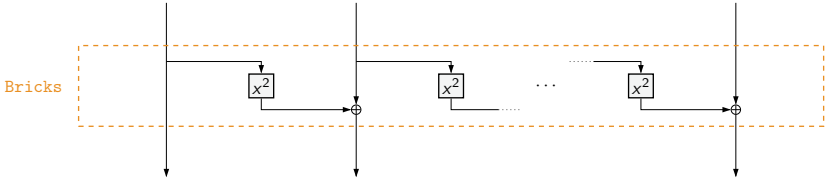
- One round (state size $t \geq 8$):



Bricks

- Non-linear layer constructed from a quadratic Feistel

$$\texttt{Bricks}(x_1, x_2, \ldots, x_t) := (x_1, x_2 + x_1^2, x_3 + x_2^2, \ldots, x_t + x_{t-1}^2).$$



- Efficient in plain and in proof systems
  - Small number of multiplications, low-degree polynomials
- Good statistical properties

Concrete

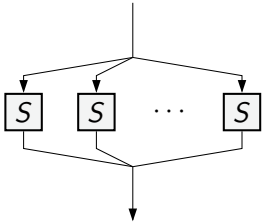
- Affine layer $M_t \cdot x + c^{(i)}$
  - ... with circular MDS matrix M_t
- Fast to compute using FFT
- Good statistical properties:
  - Optimal branch number is $t + 1$

$\Rightarrow$ Together with Bricks provides strong statistical security

$$M_8 = \begin{pmatrix} 23 & 8 & 13 & 10 & 7 & 6 & 21 & 8 \\ 8 & 23 & 8 & 13 & 10 & 7 & 6 & 21 \\ 21 & 8 & 23 & 8 & 13 & 10 & 7 & 6 \\ 6 & 21 & 8 & 23 & 8 & 13 & 10 & 7 \\ 7 & 6 & 21 & 8 & 23 & 8 & 13 & 10 \\ 10 & 7 & 6 & 21 & 8 & 23 & 8 & 13 \\ 13 & 10 & 7 & 6 & 21 & 8 & 23 & 8 \\ 8 & 13 & 10 & 7 & 6 & 21 & 8 & 23 \end{pmatrix}$$

Bars

- Binary non-linear layer
- Decompose → S-box → Compose:
  - Split into 8-bit limbs
  - SHA-3-like S-box on each limb:



$$S(x) = x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3)) \lll 1,$$

⇒ Provides strong algebraic security

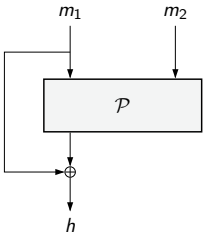
- Efficient in
  - . . . proof system due to lookup table
  - . . . plain due to fast vectorized implementation

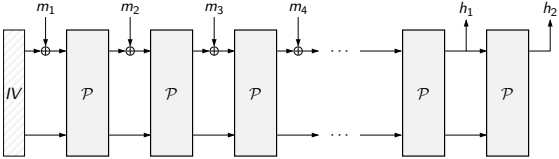
Performance

(For $p_{\texttt{Goldilocks}}$)

Modes of operation

- 2:1 compression:
  - Smaller state size is sufficient
  - E.g., for Merkle tree with fixed depth
- General-purpose hashing:
  - Sponge mode
  - Slightly larger state size





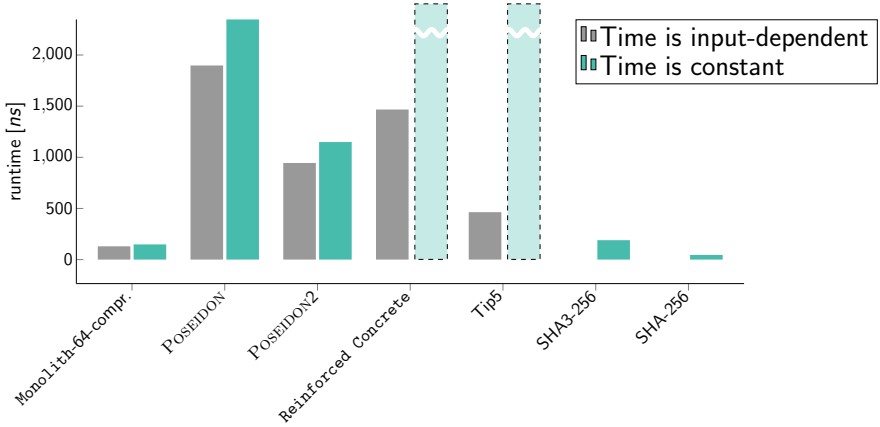
Plain Hashing Performance (Rust)

Hashing algorithm	Time for one permutation (ns) 2-to-1 compression	sponge
$p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$:	$t = 8$	$t = 12$
`Monolith`-64	**129.9**	**210.5**
Tip5 ($t = 16$)		463.6
Tip4′		247.9
Poseidon	1897.6	3288.7
Poseidon2	944.6	1291.5
Other:		
`Reinforced Concrete` (BN254)		1467.1
SHA3-256		189.8
SHA-256	45.3	

- Monolith is faster than competitors
  - Similar performance to SHA-3

Constant Time Performance

- Resisting side-channel attacks is important, even in ZK use cases
  - E.g., recently shown at Usenix by [TBP20]



Plonkish Performance: Constraining Bars

- Decomposition and binary S-box in $\mathbb{F}_p$
- Composition/decomposition constraints

  $\Rightarrow$ 8 new witnesses x_i and 8 new witnesses y_i

$$x = \sum_{i=0}^{7} x_i \cdot 2^{8i}, \quad \text{and} \quad y = \sum_{i=0}^{7} y_i \cdot 2^{8i}$$

- Lookup constraint for SHA-3-like S-box (includes range check $x_i < 2^8$ and $y_i < 2^8$)

$$(x_i, y_i) \in T[x, y], \quad \text{for } 0 \leq i < 8$$

- Equality for $x = \sum_{i=0}^{7} x_i \cdot 2^{8i} \mod p_{\texttt{Goldilocks}}$?
  - Problem: x_i which represent $x \geq p_{\texttt{Goldilocks}}$

Plonkish Performance: Constraining Bars (cont.)

- Maximum value is $p_{\texttt{Goldilocks}} - 1 = \texttt{0xFFFF FFFF 0000 0000}$
- Forbid $x_7 = x_6 = x_5 = x_4 = \texttt{0xFF}$ if any $x_i \neq \texttt{0x00}$ for $0 \leq i < 4$
- Two new witnesses z, z' and constraints

$$(x_4 2^{24} + x_3 2^{16} + x_2 2^8 + x_1) \cdot (x_8 2^{24} + x_7 2^{16} + x_6 2^8 + x_5 - z) = 0$$
$$(z - 2^{32} + 1) \cdot z' = 1$$

- Not required for output y
  - S-box ensures $S(\texttt{0x00}) = \texttt{0x00}$ and $S(\texttt{0xFF}) = \texttt{0xFF}$

$\Rightarrow$ Every valid x leads to a valid y

Plonkish Performance: Constraining Bars (cont.)

- Maximum value is $p_{\texttt{Goldilocks}} - 1 = \texttt{0xFFFF FFFF 0000 0000}$
- Forbid $x_7 = x_6 = x_5 = x_4 = \texttt{0xFF}$ if any $x_i \neq \texttt{0x00}$ for $0 \leq i < 4$
- Two new witnesses z, z' and constraints

$$(x_4 2^{24} + x_3 2^{16} + x_2 2^8 + x_1) \cdot (x_8 2^{24} + x_7 2^{16} + x_6 2^8 + x_5 - z) = 0$$
$$(z - 2^{32} + 1) \cdot z' = 1$$

- Not required for output y
  - S-box ensures $S(\texttt{0x00}) = \texttt{0x00}$ and $S(\texttt{0xFF}) = \texttt{0xFF}$

$\Rightarrow$ Every valid x leads to a valid y

Plonkish Performance: Constraining Bars (cont.)

- Maximum value is $p_{\texttt{Goldilocks}} - 1 = \texttt{0xFFFF FFFF 0000 0000}$
- Forbid $x_7 = x_6 = x_5 = x_4 = \texttt{0xFF}$ if any $x_i \neq \texttt{0x00}$ for $0 \leq i < 4$
- Two new witnesses z, z' and constraints

$$(x_4 2^{24} + x_3 2^{16} + x_2 2^8 + x_1) \cdot (x_8 2^{24} + x_7 2^{16} + x_6 2^8 + x_5 - z) = 0$$
$$(z - 2^{32} + 1) \cdot z' = 1$$

- Not required for output y
  - S-box ensures $S(\texttt{0x00}) = \texttt{0x00}$ and $S(\texttt{0xFF}) = \texttt{0xFF}$

$\Rightarrow$ Every valid x leads to a valid y

Plonkish Performance

- Plonkish arithmetization
  - Monolith's crucial advantage: Low-degree round function

Primitive	Lookups	Non-linear constraints	Degree	Witness size	Area-degree product
`Monolith-64-compression`	192	48	**2**	644	**1288**
`Monolith-64-sponge`	192	72	**2**	696	**1392**
Tip5	160	60	7	440	3080
Tip4′	160	40	7	400	2800
Poseidon/Poseidon2	0	118	7	236	1652

Benchmarks in Plonky2

- Poseidon vs. Monolith in sponge mode ($t = 12$)
  - Non-optimized custom gate using 1 row in Plonky2[1]

Primitive	Prove Time *ms*	Verify Time *ms*	Proof Size *B*	Plain Time *ns*
`Monolith-64-sponge`	3.49	0.63	112732	210.5
POSEIDON	6.23	1.12	70288	3288.7

- Results:
  - ✔ Faster prover, verifier, plain hashing, and Merkle-tree generation (15x)
  - ✖ Larger proof size

[1] Using "Implement logUp" pull request `https://github.com/mir-protocol/plonky2/pull/888`

Conclusion

- New hash function Monolith
  - Efficient in plain and in proof systems
  - Plain performance comparable to SHA-3
  - Side-channel resistant and allows constant time implementations
- Design based on two different non-linear layers
  - Bricks: $\mathbb{F}_p$ non-linear layer based on Feistel
  - Bars: Binary non-linear layer based on decomposition
- Currently fastest arithmetization friendly hash function
- Generalized description for other primes in paper

Links

- Paper:
  - https://ia.cr/2023/1025
- Plain hashing comparison:
  - https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/tree/master/plain_impls
- Plonky2:
  - https://github.com/HorizenLabs/monolith

Questions

?

Monolith:
May the Speed of SHA-3 be With You

Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger,
Christian Rechberger, Markus Schofnegger, **Roman Walch**

20.09.2023

WWW.TACEO.IO

Bibliography I

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". In: IACR Trans. Symmetric Cryptol. 2020.3 (2020), pp. 1–45.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In: EUROCRYPT (1). Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 769–793.

[GHR+22] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications". In: IACR Cryptol. ePrint Arch. (2022), p. 403.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation". In: CCS. ACM, 2022, pp. 1323–1335.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems". In: USENIX Security Symposium. USENIX Association, 2021, pp. 519–535.

Bibliography II

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. "Poseidon2: A Faster Version of the Poseidon Hash Function". In: AFRICACRYPT. Vol. 14064. Lecture Notes in Computer Science. Springer, 2023, pp. 177–203.

[Pol22] Polygon. Plonky2: Fast Recursive Arguments with PLONK and FRI. 2022. URL: https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf (visited on 04/12/2023).

[Pol23] Polygon. Plonky3. 2023. URL: https://github.com/Plonky3/Plonky3 (visited on 06/12/2023).

[Sal23] Robin Salen. "Two additional instantiations from the Tip5 hash function construction". In: https://toposware.com/paper_tip5.pdf (2023).

[SLST23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, and Bobbin Threadbare. "The Tip5 Hash Function for Recursive STARKs". In: IACR Cryptol. ePrint Arch. (2023), p. 107.

Bibliography III

[TBP20] Florian Tramèr, Dan Boneh, and Kenny Paterson. "Remote Side-Channel Attacks on Anonymous Transactions". In: USENIX Security Symposium. USENIX Association, 2020, pp. 2739–2756.