# Hash Functions `Monolith` for ZK Applications: May the Speed of SHA-3 be With You

Lorenzo Grassi[2,5], Dmitry Khovratovich[3,8], Reinhard Lüftenegger[1], Christian Rechberger[1], Markus Schofnegger[4], and Roman Walch[1,6,7]

[1] Graz University of Technology (Austria)
[2] Ponos Technology (Switzerland)
[3] Ethereum Foundation (Luxembourg)
[4] Horizen Labs (United States)
[5] Ruhr University Bochum (Germany)
[6] Know-Center (Austria)
[7] TACEO (Austria)
[8] ABDK Consulting (Estonia)

**Abstract.** The rising popularity of computational integrity protocols has led to an increased focus on efficient domain-specific hash functions, which are one of the core components in these use cases. For example, they are used for polynomial commitments or membership proofs in the context of Merkle trees. Indeed, in modern proof systems the computation of hash functions is a large part of the entire proof's complexity.

In the recent years, authors of these hash functions have focused on components which are verifiable with low-degree constraints. This led to constructions like Poseidon, *Rescue*, Griffin, `Reinforced Concrete`, and Tip5, all of which showed significant improvements compared to classical hash functions such as SHA-3 when used inside the proof systems.

In this paper, we focus on lookup-based computations, a specific component which allows to verify that a particular witness is contained in a lookup table. We work over 31-bit and 64-bit finite fields $\mathbb{F}_p$, both of which are used in various modern proof systems today and allow for fast implementations. We propose a new 2-to-1 compression function and a SAFE hash function, instantiated by the `Monolith` permutation. The permutation is significantly more efficient than its competitors, both in terms of circuit friendliness and plain performance, which has become one of the main bottlenecks in various use cases. This includes `Reinforced Concrete` and Tip5, the first two hash functions using lookup computations internally. Moreover, in `Monolith` we instantiate the lookup tables as functions defined over $\mathbb{F}_2$ while ensuring that the outputs are still elements in $\mathbb{F}_p$. Contrary to `Reinforced Concrete` and Tip5, this approach allows efficient constant-time plain implementations which mitigates the risk of side-channel attacks potentially affecting competing lookup-based designs. Concretely, our constant time 2-to-1 compression function is faster than a constant time version of Poseidon2 by a factor of 7. Finally, it is also the first arithmetization-oriented function with a plain performance comparable to SHA3-256, essentially closing the performance gap between circuit-friendly hash functions and traditional ones.

# Table of Contents

# 1 Introduction

## 1.1 Hash Functions in Zero-Knowledge Frameworks

Zero-knowledge use cases and particularly the area of computational integrity combined with zero knowledge have seen a rise in popularity in the last couple of years. Many new protocols and low-level primitives (such as optimized hash functions) have been designed and published recently [GWC19; ZGK+22; GHR+22; GKL+22], in an attempt to increase the performance in this setting. These improvements have especially been pushed by modern blockchain implementations, where the validity of a specific state can be proven significantly more efficiently when employing zero-knowledge proofs (see, e.g., [SC21]). With the emergence of parallel proving techniques and recursive SNARKs (*incrementally verifiable computation*, or IVC) it has become possible to efficiently prove the integrity of complex computations. Proofs with $2^{27}$ steps have been recorded[1] whereas Ethereum Foundation is planning to prove up to $2^{40}$ steps in its VDF hardware implementation [KMT22].

With VC programs (also called circuits) being that large and containing cryptographic protocols, more and more programs contain hash functions as subroutines. Hash functions and their underlying permutations are not only components for plain hashing, but also of commitment schemes, authenticated encryption, Fiat–Shamir conversions, and other concepts. A notable case is the use of a hash function to build a Merkle-based commitment [BBH+19] as part of a recursive SNARK, which is then opened at the next step of the recursion to prove correctness [COS20]. A hash function in this case must be both fast enough in plain to efficiently compute a Merkle tree and small enough to be part of a circuit defined over some prime field.

All these use cases can be accomplished with classical and well-analyzed hash functions such as SHA-3 [Nat15]. However, the performance of the underlying proving schemes heavily depends on certain properties which are not fulfilled by them. This performance gap led to many new designs, dubbed *arithmetization-oriented* or *circuit-friendly* hash functions due to their efficient arithmetic representations. While some of them focus on round functions which are exclusively built from low-degree components (Poseidon [GKR+21], Poseidon2 [GKS23], Neptune [GOP+22], MiMC/GMiMC-like functions [AGR+16; AGP+19]), others focus on high-degree round functions which allow for equivalent low-degree representations in the proof system (Friday [AD18], *Rescue* [AAE+20; SAD20], Griffin [GHR+22], Anemoi [BBC+22]).

## 1.2 First Lookup-Friendly Designs

Poseidon and its relatives are significantly more efficient in proof systems compared to, e.g., SHA-3. However, their plain performance is usually several orders of magnitude slower. This problem is addressed with a new line of research focusing on lookup-based hash functions. These are built from high-degree components which can efficiently be implemented as a lookup table, significantly improving the plain performance. While this may lead to a large number of constraints in some proof systems, the performance in proof systems supporting lookup arguments [GW20] is still on par with other arithmetization-oriented hash functions. Examples for these new designs are `Reinforced Concrete` [GKL+22] and the recent Tip5 [SLS+23].

The round functions of both `Reinforced Concrete` and Tip5 generally follow a classical structure, containing an affine layer (which in both cases is a simple matrix multiplication followed by a round constant addition) and a nonlinear layer. The latter is instantiated with the `Horst` approach [GHR+22] in the case of `Reinforced Concrete`. In Tip5, one part of the linear layer consists of monomial power functions $x \mapsto x^d$ for a small $d$, while the other part (applied to four state elements) consists of high-degree functions using lookup tables internally.

`Reinforced Concrete` was the first circuit-friendly hash function specifically designed to increase the plain performance by using lookup arguments in the proofs. Its security analysis involves the novel component `Bars`, which adds a significant amount of algebraic complexity by essentially changing the domain over which the computation takes place. This complex structure allows for strong arguments against algebraic attacks, comparable to those made for AES.

[1] https://research.protocol.ai/sites/snarks/

### 1.3 Performance Gains in Smaller Fields

The domain over which a verifiable computation is defined is fully determined by the proof system. The zero-knowledge proof systems such as Groth16 [Gro16] and Plonk [GWC19] rely on pairings and their computation domains are necessarily scalar fields of pairing-secure elliptic curves. These scalar fields are about 256 bits large for providing 128 bits of security. In contrast, proof systems relying on FRI commitment schemes, though producing larger proofs, can operate on smaller fields. Using smaller prime fields has several advantages in the FRI setting. For example, the trace elements become smaller, leading to a slimmer representation overall. Moreover, while FFT computations take the same amount of operations for traces of the same length regardless of the field size, they can be evaluated faster when smaller fields with more efficient arithmetic operations are used. Examples are fields based on prime numbers $2^{64} - 2^{32} + 1$ and $2^{31} - 1$. Both of them allow for efficient modular reductions (e.g., see Appendix A) and have been chosen in various recent proving frameworks such as Plonky2 [Pol22a], its successor Plonky3 [Pol23], and Risc0 [RIS23]. There are also various works in recent literature discussing the use of smaller prime fields in this context [HLN23; Hab23].

### 1.4 Shortcomings of `Reinforced Concrete` and Tip5

Both designs have various notable shortcomings.

- *Performance:* There is still a noticeable performance gap to SHA-3 (Table 5). Recursive FRI-based ZK schemes such as Fractal [COS20] need the same hash function to constitute a Merkle tree and to be proven in circuits. As a result, the generation of Merkle tree commitments takes up to half of the overall time required for a proof [Pol22b]. Therefore, improving the plain performance of hash functions is a promising optimization target.
- *Side Channels:* The usage of lookup tables is a well known source of side channel leakage. Whenever a secret information is processed, an adversary may recover a large portion of it from timing differences of lookups into memory or caches. These works are well known since at least two decades in the context of encryption [Pag02; Ber05; OST06], and have recently applied to zero-knowledge proof systems [TBP20]. As tables in `Reinforced Concrete` and Tip5 are rather big, it is non-trivial to have a constant-time implementation with reasonable overhead. The natural ways to convert a table to a polynomial or a bitsliced implementation result in a large amount of multiplications and thus significantly worse performance. This also implies that implementing such a function in a legacy proof system with no lookups would result in a significant slowdown.
- *Decomposition:* In order to apply lookup-based functions over a fewer number of bits, the original (larger) field element is usually split into smaller elements from a different field. This step is often called *decomposition*. Specifically for `Reinforced Concrete`, the decomposition of a field element into smaller table inputs requires a chain of modular reductions into smaller prime fields. As a result, it is inherently slow and a major bottleneck in the computation.

### 1.5 `Monolith`: High-Speed, Constant-Time-Friendly, ZK-Oriented Hashing

Our main contribution is `Monolith`, a family permutations which are both efficient in plain and inside of circuits and can be turned into hash functions and other permutation-based schemes. We first present the key ideas behind the new design and then explain how we build the components of `Monolith` using these ideas.

**Main `Monolith`'s Components: Combination of New Ideas and Well-Known Ones.** We take inspiration from the components in `Reinforced Concrete` and Tip5 and augment them with novel techniques to address the shortcomings from the previous section. Note that the small prime field, while being advantageous for arithmetic performance, requires a large number of words in the state and thus creates performance bottlenecks in diffusion layers. Our answers to those challenges are as follows.

– *Low-Degree Nonlinear Functions:* Instead of using components such as $x \mapsto x^d$ for a small $d > 2$, we use only quadratic mappings in the nonlinear layers. This reduces the number of constraints in the circuit and at the same time requires fewer modular reductions on a CPU.
– *Lookup-Friendly and Constant-Time-Friendly S-Boxes:* We define our lookup table over a binary extension field of sizes 7 and 8 bits, instantiated by Daemen's $\chi$ function or similar ones [Dae95]. These can be implemented using fast vector instructions and allow for the parallel evaluation of multiple S-boxes.
– *Flexible Design:* While we target specific prime fields and security levels for the ease of implementation and analysis, all our components and their analysis are mostly independent of the underlying prime field and scale both to bigger and smaller fields.
– *Special Matrices for the Linear Layer:* The large state requires matrices of size up to $24 \times 24$, where the quadratic cost of matrix multiplication is significant. We reuse circulant MDS matrices from Tip5 and the Winterfell library, which allow for fast multiplications using NTT.[2]

**Circuit-Friendly Permutations for Small Primes.** On the high level, the new permutations follow the design of `Reinforced Concrete` and to some extent Tip5, which themselves are built upon the well-known SPN paradigm. Our scheme consists of a few rounds, each using the following three components.

– The first one is `Bricks` (Section 2.4), which uses low-degree nonlinear functions and is built on a Feistel Type-3 construction [ZMI89]. It consists of square mappings and provides resistance against statistical attacks such as differential ones.
– The second component is `Bars` (Section 2.6), where field elements are decomposed into smaller chunks and $\chi$-based S-boxes are applied to them. We prove that each such `Bar` operation has a high degree and provide a security analysis against algebraic attacks. From our results, it is sufficient to apply `Bar` only to a few field elements in each round.
– Finally, the third component is `Concrete` (Section 2.5), which is the multiplication with a circulant MDS matrix. It provides diffusion and is necessary to gain security against statistical attacks.

The combination of these three components in each round contributes to the security against statistical and algebraic attacks while allowing for an efficient implementation. Our initial analysis suggests the possibility to set up attacks on up to 4 rounds of Monolith. Since improvements are expected (we encourage third-party cryptanalysis), we set the number of rounds uniformly to 6.

**Performance Evaluation.** We give an extensive comparison between our new proposal and its competitors in Section 6. Our benchmarks confirm that the plain performance of `Monolith` in software is comparable to SHA-3, which makes it the first circuit-friendly compression function achieving this goal. At the same time, `Monolith` is efficient in combination with zero-knowledge proof systems. In contrast to `Reinforced Concrete` and Tip5, `Monolith` also has the crucial advantage that it allows for a constant-time implementation without significant performance losses (see Fig. 1), and it can also be reasonably used in proof systems without lookup arguments.

Further, compared to Tip5, `Monolith` is around twice as fast and gives the user more freedom regarding the choice of the prime number. Indeed, it can even be used with prime fields as low as 31 bits, which is a setting recently considered in the literature and various proving frameworks due to advantageous implementation characteristics. Moreover, compared to the widely used Poseidon permutation, `Monolith` shows a plain performance improvement by a factor of around 15. Finally, `Monolith` allows for an efficient circuit implementation, since it can be represented by a low number of degree-2 constraints.

**Security Analysis Summary.** We have conducted extensive analysis of our design in the context of various attacks (Section 5). As some of the components or combinations are new, our analysis contains several non-trivial ideas and may be of separate interest to cryptanalysts and designers. Here are several insights.

[2] https://github.com/facebook/winterfell/

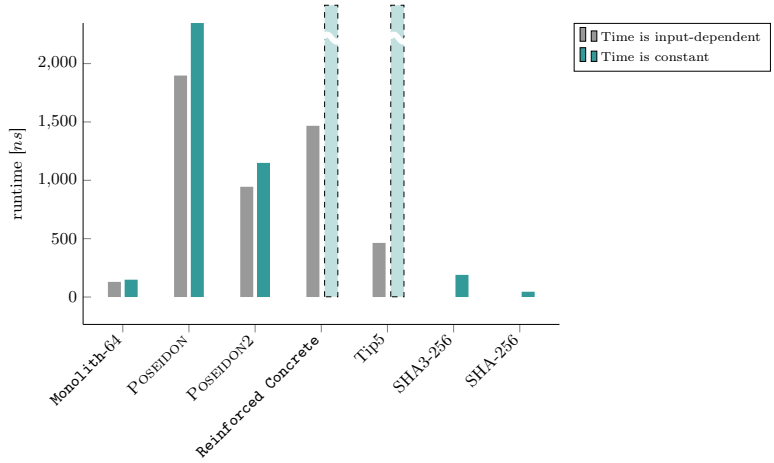


**Fig. 1.** Runtime comparison of different hash functions. The benchmarks are from Table 5 and Table 6 and the numbers for Monolith-64, Poseidon, and Poseidon2 are taken for the 64-bit prime field and a state size of $t = 8$. (We acknowledge that the comparison is between 2-to-1 compression functions and sponge hash functions.)

- In the spirit of the wide trail strategy [DR02], we prove tight bounds for the number of active squarings in differential characteristics for the Type-3 Feistel-MDS combination in Section 5.1.
- We study rebound attacks in Section 5.2, a research direction that is often missed in the ZK hash function design. We demonstrate practical attacks on a reduced version of Monolith and argue the security of the full version.
- Using differential and linear properties of Bar, we prove lower bounds on its algebraic degree in Section 5.3, which imply resistance against algebraic attacks after a few rounds.
- While arguing the security of Monolith against algebraic attacks, we study the complexity of Gröbner basis attacks on toy versions of Monolith with smaller primes but still realistic Bars layers in Section 5.4.

## 2 Specification of Monolith

In the following, we define the symmetric primitives Monolith-64 and Monolith-31. Monolith-64 is defined over $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$, whereas Monolith-31 is defined over $p_{\texttt{Mersenne}} = 2^{31} - 1$. The main difference lies in the definition of the Bars layer. For this reason, we first describe Monolith-64 and Monolith-31 in a generic matter, using the "Monolith" notation, and we then give the specification for our schemes.

### 2.1 Domain

Our main two instances work over $\mathbb{F}_p^t$, where $p$ is either $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$ or $p_{\texttt{Mersenne}} = 2^{31} - 1$, and $8 \leq t \leq 24$. The instances using $p_{\texttt{Goldilocks}}$ provide a security level of $2\log_2(p_{\texttt{Goldilocks}}) \approx 128$ bits and the instances using $p_{\texttt{Mersenne}}$ provide a security level of $4\log_2(p_{\texttt{Mersenne}}) \approx 124$ bits.

### 2.2 Modes of Operation

We suggest two modes of operation for Monolith, an arbitrary-length hashing one and a fixed-size compression one. The former is useful for general-purpose hashing or processing the leaf data in a Merkle tree, whereas the latter is useful for fixed compression ratios (such as e.g. 2-to-1) in the upper levels of a Merkle tree construction.

**SAFE-Based Schemes.** A `Monolith` permutation can be plugged into the SAFE sponge framework [AKM+22] and implement variable-length hash functions, commitment schemes, authenticated encryption, stream ciphers, and other schemes. The SAFE framework is an extension of earlier duplex and sponge constructions [BDP+07; BDP+08], where the permutation state is split into an outer part with a rate of $r$ elements and an inner part with a capacity of $c$ elements. A crucial difference to a classical sponge framework (without any modifications) is that SAFE handles domain separation automatically by initializing the capacity part with a specific value derived from the input-output pattern of the scheme. In particular, the sequence of the input and the output lengths is used to separate the different applications of the underlying permutation.

For a security level of $\kappa$ bits, we require that $c \geq \left\lceil \frac{2\kappa}{\log_2(p)} \right\rceil$ and that the output of the hash function consists of at least $2\kappa/\log_2(p)$ elements. For example, for a 64-bit prime field, we suggest $r = 8$ and $c = 4$ (hence, a state size of $t = 8 + 4 = 12$) to obtain a security level of 128 bits, while requiring only a single permutation call to process two 256-bit inputs (each one requiring four elements in the rate part).

**2-to-1 Compression Function.** We also suggest a fixed-length $t$-to-$n$ *compression function*. Concretely, it takes $t$ $\mathbb{F}_p$ elements as input and produces $n$ $\mathbb{F}_p$ elements as output. It is defined as

$$x \in \mathbb{F}_p^t \mapsto \mathcal{Z}(x) := \mathrm{Tr}_n(\mathcal{P}(x) + x) \in \mathbb{F}_p^n,$$

where $\mathrm{Tr}_n$ yields the first $n$ elements of the inputs. A compression function can be used in Merkle trees with various arities and has recently also been applied in similar constructions, including Anemoi [BBC+22], Griffin [GHR+22], and Poseidon2 [GKS23].

For a security level of $\kappa$ bits and assuming a pseudo-random (known) permutation for $\mathcal{P}$, $\mathcal{Z}$ is a secure compression function with respect to collisions and (second-)preimages if $p^n \geq 2^{2\kappa}$ (due to the birthday bound attack) and $p^{t-n} \geq 2^\kappa$ (in order to avoid a guessing attack on the truncated part). For the goal of this paper, we limit ourselves to the case $t = 2n$. Taking the example from above with a 64-bit prime field, we suggest $t = 8$ and $n = 4$ to obtain a security level of 128 bits, while requiring only a single permutation call to process two 256-bit inputs (each one requiring four elements of the input).

### 2.3 Permutation Structure

The `Monolith` permutation is defined as

$$\texttt{Monolith}(\cdot) = \mathcal{R}_r \circ \cdots \circ \mathcal{R}_2 \circ \mathcal{R}_1 \circ \mathcal{R}'(\cdot),$$

where $r$ is the number of rounds and $\mathcal{R}_i, \mathcal{R}'$ over $\mathbb{F}_p^t$ are defined as

$$\begin{aligned} \mathcal{R}'(\cdot) &= \texttt{Concrete}(\cdot)\,, \\ \mathcal{R}_i(\cdot) &= c^{(i)} + \texttt{Concrete} \circ \texttt{Bricks} \circ \texttt{Bars}(\cdot), \qquad \forall i \in \{1, 2, \ldots, r\}\,, \end{aligned}$$

where `Concrete` is a linear operation, `Bars` and `Bricks` are nonlinear operations over $\mathbb{F}_p^t$ and $c^{(1)}, \ldots, c^{(r-1)} \in \mathbb{F}_p^t$ are pseudo-random round constant. The last layer of round constants $c^{(r)}$ is set to $\mathbf{0}$. Note that a single `Concrete` operation is applied before the first round. A graphical overview of one round of the construction is shown in Fig. 2.

### 2.4 `Bricks`

The component `Bricks` over $\mathbb{F}_p^t$ is defined as a Feistel Type-3 construction [ZMI89] (without shift) instantiated with a square map $x \mapsto x^2$, i.e.,

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

where we denote by $x_i$ the $i$-th entry of the vector $x \in \mathbb{F}_p^t$.

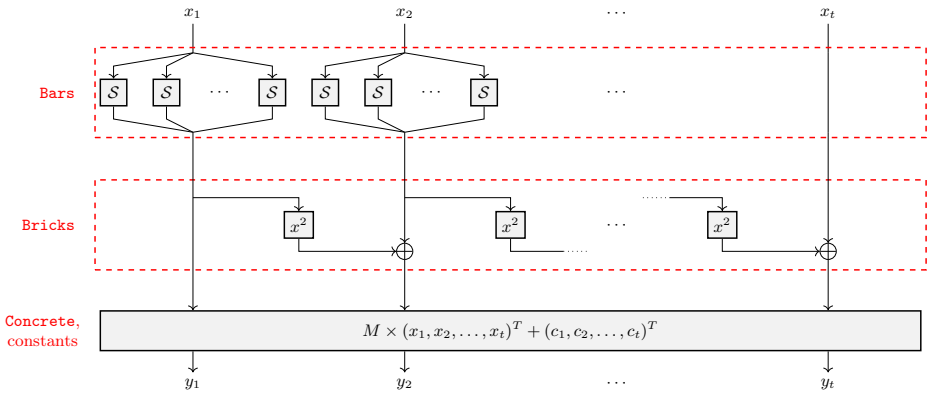


**Fig. 2.** One round of the `Monolith` construction, where $x_i, y_i \in \mathbb{F}_p$.

### 2.5 `Concrete`

The `Concrete` layer serves as a linear layer to achieve strong diffusion required for statistical security. In particular, `Concrete` is defined as

$$\texttt{Concrete}(x_1, x_2, \ldots, x_t) := M \times (x_1, x_2, \ldots, x_t)^T,$$

where $M \in \mathbb{F}_p^{t \times t}$ is an MDS matrix. Since the multiplication with an MDS matrix is in general expensive and requires a number of operations in $\mathcal{O}(t^2)$, we use matrices with special properties.

- **Goldilocks Prime** $p_{\texttt{Goldilocks}}$**.** We use the circulant matrix $\text{circ}(23, 8, 13, 10, 7, 6, 21, 8)$ for $t = 8$ and the matrix $\text{circ}(7, 23, 8, 26, 13, 10, 9, 7, 6, 22, 21, 8)$ for $t = 12$, as found and implemented by the Winterfell STARK library.[3] These matrices have the unique advantage of having small elements in the time and frequency domain (i.e., before and after DFT application), allowing for especially fast plain performance.
- **Mersenne Prime** $p_{\texttt{Mersenne}}$**.** We instantiate $M$ via the matrix used in Tip5 [SLS+23] for $t = 16$, since it is also MDS for $p_{\texttt{Mersenne}}$.[4] Since we are not aware of any fast MDS matrix for $t = 24$, we suggest to use a random Cauchy matrix [YMT97] in the concrete layer at the cost of a slower plain performance. The problem of finding a fast MDS matrix for this larger state size (which would significantly increase the plain performance of `Monolith` $-$ 31 with $t = 24$) is left as future work.

### 2.6 `Bars`

The `Bars` layer is defined as

$$\texttt{Bars}(x_1, x_2, \ldots, x_t) := \texttt{Bar}(x_1) \,||\, \texttt{Bar}(x_2) \,||\, \cdots \,||\, \texttt{Bar}(x_u) \,||\, x_{u+1} \,||\, \cdots \,||\, x_t \tag{1}$$

for a $t$-element state, where $u \in \{1, \ldots, t\}$ denotes the number of `Bar` applications in a single round. Each `Bar` application is defined as

$$\texttt{Bar}(x) = \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x), \tag{2}$$

where

- the *decomposition* $\mathcal{D} : \mathbb{F}_p \to \mathbb{F}_2^{n_1} \times \mathbb{F}_2^{n_2} \times \cdots \times \mathbb{F}_2^{n_m}$ reads the original field element $x \in \mathbb{F}_p$ as an integer and splits it into $m$ chunks $y_1, y_2, \ldots, y_m$,
- $\mathcal{S}$ is the parallel application of $m \geq 2$ bijective S-boxes over $\mathbb{F}_2^{n_i}$ such that

$$\mathcal{S}(y_1, y_2, \ldots, y_m) = \mathcal{S}_1(y_1) \,||\, \mathcal{S}_2(y_2) \,||\, \cdots \,||\, \mathcal{S}_m(y_m), \tag{3}$$

[3] https://github.com/facebook/winterfell/tree/main/crypto/src/hash/mds

[4] https://github.com/Neptune-Crypto/twenty-first/blob/master/twenty-first/src/shared_math/tip5.rs

– the *composition* $\mathcal{C}$ is the inverse operation of the decomposition.

We now describe these components individually for both `Monolith-64` and `Monolith-31`. For a more generic description of `Bars` and `Bar` and for the reasoning behind the building blocks we refer to Section 4.

**`Bars` for `Monolith-64`.** In Eq. (1) we set $t \in \{8, 12\}$ (compression or sponge use case, resp.) and we set $u = 4$ (i.e., 4 `Bar` operations are applied in each round).

*Decomposition $\mathcal{D}$ and Composition $\mathcal{C}$.* We use a decomposition into 8-bit values s.t.

$$x = 2^{56}y_8 + 2^{48}y_7 + 2^{40}y_6 + 2^{32}y_5 + 2^{24}y_4 + 2^{16}y_3 + 2^8y_2 + y_1.$$

The composition $\mathcal{C}$ is the inverse operation of the decomposition.

*S-Boxes $\mathcal{S}$.* In Eq. (3) we set $m = 8$. Then all $\mathcal{S}_i$ over $\mathbb{F}_2^8$ are identical as (see [Dae95, Table A.1])

$$\mathcal{S}_i(y) = \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \tag{4}$$

where $\lll$ is a circular shift (here we interpret an integer as a big-endian 8-bit string) and $\overline{y}$ is the bitwise negation.

**`Bars` for `Monolith-31`.** In Eq. (1) we set $t \in \{16, 24\}$ (compression or sponge use case, resp.) and we set $u = 8$ (i.e., 8 `Bar` operations are applied in each round).

*Decomposition $\mathcal{D}$ and Composition $\mathcal{C}$.* The decomposition $\mathcal{D}$ is given by

$$x = 2^{24}y'_4 + 2^{16}y_3 + 2^8y_2 + y_1,$$

where $y'_4 \in \mathbb{Z}_2^7$ and $y_3, y_2, y_1 \in \mathbb{Z}_2^8$. The composition $\mathcal{C}$ is the inverse operation of the decomposition.

*S-Boxes $\mathcal{S}$.* In Eq. (3) we set $m = 4$ using $\{8, 7\}$-bit lookup tables. Then the S-boxes are defined as (see [Dae95, Table A.1])

$$\begin{aligned} \forall i \in \{1, 2, \ldots, m-1\}: \quad \mathcal{S}_i(y) &= \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \\ \mathcal{S}_m(y') &= \big(y' \oplus \big((\overline{y'} \lll 1) \odot (y' \lll 2)\big)\big) \lll 1, \end{aligned} \tag{5}$$

where $y \in \mathbb{F}_2^8$ and $y' \in \mathbb{F}_2^7$.

### 2.7 Round Constant Generation

The round constants $c_1^{(i)}, c_2^{(i)}, \ldots, c_t^{(i)}$ for the $i$-th round are generated using the well-known approach of seeding a pseudo-random number generator and reading its output stream. In particular, we use SHAKE-128 with rejection sampling, i.e., we discard elements which are not in $\mathbb{F}_p$. SHAKE-128, thereby, is seeded with the initial seed "`Monolith`" followed by the state size $t$ and number of rounds $r$, each represented as one byte, the prime $p$ represented by $\lceil \log_2(p)/8 \rceil$ bytes in little endian representation, and the decomposition sizes in the bar layer, where each $s_i$ is represented as one byte. Thus, the seed is

```
b'Monolith\x08\x06\x01\x00\x00\x00\xff\xff\xff\xff\x08\x08\x08\x08\x08\x08\x08\x08'
```

for `Monolith-64` with $t = 8, r = 6$ and

```
b'Monolith\x10\x06\xff\xff\xff\x7f\x08\x08\x08\x07'
```

for `Monolith-31` with $t = 16, r = 6$.

**Table 1.** Parameters for `Monolith`.

| Name | $p$ | Security ($\kappa$ bits) | Rounds $r$ | Width $t$ | | # `Bar` $u$ |
|---|---|---|---|---|---|---|
| | | | | Compression | Sponge | |
| `Monolith-64` | $2^{64} - 2^{32} + 1$ | 128 | 6 | 8 | 12 | 4 |
| `Monolith-31` | $2^{31} - 1$ | 124 | 6 | 16 | 24 | 8 |

### 2.8 Number of Rounds

In Table 1, we propose to use $r = 6$ rounds for `Monolith-64` and `Monolith-31`, which comes with security claims of $2\log_2(p_{\texttt{Goldilocks}}) \approx 128$ bits and $4\log_2(p_{\texttt{Mersenne}}) \approx 124$ bits, respectively. These numbers are conservatively chosen based on the security analysis proposed in Section 5. We note that we do not use the `Bars` layer in the analysis against statistical attacks, and we do not use the `Bricks` layer to argue algebraic security. Consequently, these layers act as an additional security margin of the design.

## 3 Design Rationale

We carefully chose the components `Bricks`, `Concrete`, and `Bars` in order to provide specific properties outlined below. In particular, we focused on the best plain performance while not impacting the efficiency in the proof systems.

### 3.1 Starting Point: `Reinforced Concrete`

Proposed in 2022 [GKL+22], `Reinforced Concrete` is arguably the first lookup-based circuit-friendly hash function. It has shown that the performance advantage compared to other approaches in this setting can be significant. Thus, it was a natural choice to use `Reinforced Concrete` as a basis when designing our new construction for smaller prime fields. However, some of these building blocks are relatively expensive. For instance, the decomposition requires a chain of modular reductions into smaller prime fields which are not CPU-friendly. Our plan was to modify the components of `Reinforced Concrete` in order to tailor them to certain prime fields and make the resulting design even faster.

We follow the naming convention established in `Reinforced Concrete` with the main components being called `Concrete`, `Bricks`, and `Bars`. While `Concrete` and `Bricks` operate over large field elements ($\approx$ 256 bits in the case of `Reinforced Concrete`), `Bars` decomposes them into elements of a smaller prime field and applies small parallel S-boxes to the buckets, composing back thereafter. These S-boxes are implemented as a lookup table, which is also used in the plain implementation despite not being constant-time.

An graphical comparison between `Reinforced Concrete` and our new permutation `Monolith` is shown in Fig. 3.

### 3.2 Structure of a Round

In contrast to `Reinforced Concrete`, a single `Bars` layer is not sufficient to prevent algebraic attacks. This is based on the fact that the maximum degree of each `Bars` is only $p$ ($\approx 2^{64}$ or $\approx 2^{31}$ for the instances considered in this paper). This implies that we need at least two (four for $p \approx 2^{31}$) of these operations in order to reach the desired degree of $\approx 2^{128}$.

However, we also need diffusion between these `Bars` layers, which is why we decided to incorporate `Bars` with `Concrete` and `Bricks` in a single new round function such that

$$x \mapsto c + \texttt{Concrete} \circ \texttt{Bricks} \circ \texttt{Bars}(x).$$

Iterating a single round function also has advantages compared to Poseidon or in general Hades-like schemes, where two different round functions have to be implemented in the circuit.

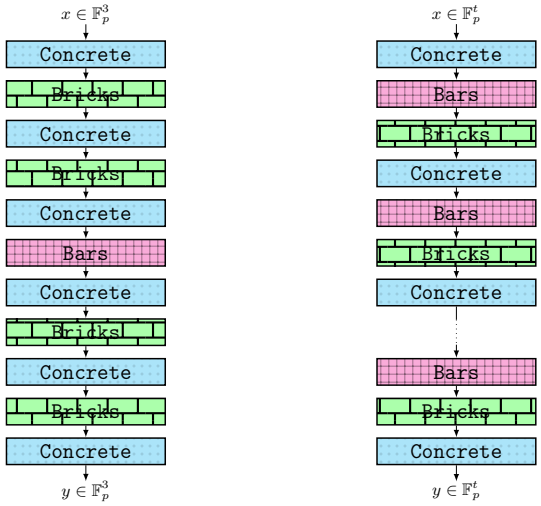


**Fig. 3.** Structure of the `Reinforced Concrete` (left) and the `Monolith` (right) permutations, where we omit the round constant additions for simplicity and $t \in \{8, 12, 16, 24\}$ for `Monolith`.

### 3.3 The `Bricks` Layer

To achieve an appropriate level of statistical security, we require that the maximum differential probability of a nonlinear operation is sufficiently low. A natural way to achieve a low upper bound for that is to use a low-degree nonlinear function, which has been done in many of the recent arithmetization-oriented designs, including Poseidon and similar constructions. However, since many of them are SPNs, every function in the nonlinear layer needs to be invertible. For a function defined by the power map $x \mapsto x^d$, this means that $\gcd(p, d-1) = 1$ in $\mathbb{F}_p$, which implies that $d \geq 2$ and in particular $d = 7$ for $p = 2^{64} - 2^{32} + 1$. This increases both the number of nonlinear operations (and hence modular reductions) in the implementation and the number of quadratic constraints.

Moving to a degree-2 nonlinear function appears to be the better approach, since it achieves the lowest bound for the maximum differential probability and the lowest number of multiplications and constraints required. Since degree-2 functions are not invertible over $\mathbb{F}_p$, we use them in a scheme like the Feistel Type-3 (chosen over, e.g., Type-2 to maximize the number of square maps) which is invertible independently of the details of the round function. In particular, the maximum differential probability of the squaring operation is $1/p$. This allows us to use only a small number of `Bricks` layers while still obtaining the advertised level of security. Moreover, the nonlinear squaring operation provides strong inner-word diffusion, almost optimal nonlinearity, good plain performance characteristics, and can be efficiently represented in a circuit.

### 3.4 The `Concrete` Layer

The `Concrete` layer consists of a dense matrix and is used for full diffusion over the entire state after a single round. Omitting it would require multiple rounds to get better diffusion, since otherwise element-wise diffusion is only provided by the `Bricks` layer. This also allows us to omit the Feistel swap from the `Bricks` layer, as proposed alread in, e.g., [BFM+16].

Moreover, we chose an MDS matrix, i.e., we benefit from a (maximum) branch number of $t+1$ for a $t$-element state. This results in strong statistical properties in combination with `Bricks`. Moreover, for all $t \leq 16$, we focus on circulant MDS matrices which allow us using FFT in order to compute the matrix multiplication efficiently.

*Full Diffusion at the Input and the Output.* In order to establish immediate diffusion and to prevent skipping nonlinear layers in the CICO attacks shown in [BBL+22], we apply `Concrete` layers at the beginning and at the end of `Monolith`.

### 3.5 The Bars Layer

We designed Bars with efficiency and algebraic strength in mind, focusing on the following properties.

- Bars should be fast on modern CPUs and can benefit from vector instructions.
- Bars should allow constant-time instructions with minimal or even no overhead.
- Bars and its inverse should have a high algebraic degree.
- Bars and its inverse should be described by dense polynomials over $\mathbb{F}_p$.
- Bars can be decomposed into operations that can be efficiently implemented in zero-knowledge circuits supporting lookup arguments.

The form of the prime $p$ makes a decomposition easier compared to Reinforced Concrete, where an element of $\mathbb{F}_p$ is represented as a vector from $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \mathbb{Z}_{p_l}$, which makes the decomposition expensive both natively and inside the proof system. In contrast, for primes of the form $2^n - 1$ and $2^n - 2^\eta + 1$ we can use the decomposition into elements of $\mathbb{Z}_{2^k}$, i.e., bit chunks. The bijectivity is guaranteed by requiring that S-boxes have fixed points at 0 and $-1$ (all bits are ones). Hence, our design is forward-compatible with other primes of the same form.

The chosen 8-bit and 7-bit S-boxes can be efficiently implemented with vector instructions (our fastest implementation requires only $10\,ns$ for the full layer for $t = 8$), so that we do not need lookup tables in the native implementation. This can also be seen from the representation given in Eq. (6) and is again a sharp difference to Reinforced Concrete and Tip5, where the S-box does not admit a simple representation and lookups are therefore required also for an efficient plain implementation.

For the form of the S-box, we were looking for fast nonlinear operations. The $\chi$ function from Keccak [Nat15] and its relatives is a natural choice. Whereas $\chi$ is not invertible for an even number of bits, our selection is one of the few which are.

Finally, note that the functions we apply in Bars result in high-degree functions over $\mathbb{F}_p$ as shown in Proposition 2. We also have strong evidence that the polynomials representing Bars applied to a single field element are dense (we refer to Section 5.3 for our experimental results). This is crucial, since the other components in Monolith are low-degree functions, and hence we need Bars to reach the maximum degree of the permutation and provide security against algebraic attacks.

## 4 Analysis of Bar

One of the core components for making our new hash function efficient without using lookup tables is an efficient representation of the lookup operation over $\mathbb{F}_2$. While this is a prominent and well-known topic in the area of cryptographic primitives optimized for software and especially hardware implementations, to the best of our knowledge this approach has never been considered for primitives over large prime fields.

Choosing this approach has one crucial advantage. While the performance is similar to a lookup-based approach, we can implement the lookup operation with a constant number of efficient instructions. Moreover, taking inspiration from existing primitives over $\mathbb{F}_2$ such as Keccak, we show how to implement multiple lookup operations in parallel, further minimizing the number of instructions we need.

In this section we propose a generic component called Bar, which is one of the main components we use in Monolith. Here we give a generic description of the Bar component (see Eq. (2)), described in Section 2.6 for Monolith-64 and Monolith-31. We also prove its invertibility and well-definition. Our description is generic for the prime fields $\mathbb{F}_p$ with $p$ being either of the form $p_{\texttt{gen1}} = 2^n - 2^\eta + 1$ or of the form $p_{\texttt{gen2}} = 2^n - 1$.

*Additional Notation.* In order to define the lookup operations, we denote the sizes (in bits) of the lookup tables by $s_1, s_2, \ldots, s_m \geq 1$ such that

$$s_1 + s_2 + \cdots + s_m = n = \lceil \log_2(p) \rceil \,,$$

where $p \in \{p_{\texttt{gen1}}, p_{\texttt{gen2}}\}$. If $p = p_{\texttt{gen1}}$, we additionally require that there exists $l \in \{1, 2, \ldots, m-1\}$ such that

$$s_1 + s_2 + \cdots + s_l = \eta \quad \text{and} \quad s_{l+1} + s_{l+2} + \cdots + s_m = n - \eta.$$

For efficiency reasons, we suggest to choose $s_1 = s_2 = \cdots = s_m = n/m$ whenever possible. Otherwise, we suggest to choose $s_1, s_2, \ldots, s_m$ to be integers close to $n/m$ that (i) satisfy the previous requirements and (ii) for which $\max_{i \neq j}\{|s_i - s_j|\}$ is minimized. Note that the parameters chosen for `Monolith`-64 and `Monolith`-31 fulfill these criteria.

### 4.1 Decomposition $\mathcal{D}$

The decomposition $\mathcal{D}$ decomposes the original field element $x \in \mathbb{F}_p$ into $m > 1$ smaller elements $x_1, x_2, \ldots, x_m$, where $x_i \in \mathbb{Z}_2^{s_i} \equiv \mathbb{F}_2^{s_i}$. For this purpose, we first set

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} \cdot x_i'$$

for $x_i' \in \mathbb{Z}_{2^{s_i}}$. Then, for each $i \in \{1, 2, \ldots, m\}$, $x_i \in \mathbb{Z}_2^{s_i}$ is the binary decomposition of $x_i' \in \mathbb{Z}_{2^{s_i}}$.

### 4.2 S-Boxes $\mathcal{S}$

The operation $\mathcal{S}$ is the parallel application of $m$ S-boxes, i.e.,

$$\mathcal{S}(x_1, x_2, \ldots, x_m) = \mathcal{S}_1(x_1) \,||\, \mathcal{S}_2(x_2) \,||\, \cdots \,||\, \mathcal{S}_m(x_m),$$

where $\mathcal{S}_i : \mathbb{F}_2^{s_i} \to \mathbb{F}_2^{s_i}$. To guarantee that `Bar` is well-defined, we require that `0b11..11` $\in \mathbb{F}_2^{s_i}$ is a fixed point, i.e.,

$$\mathcal{S}_i(\texttt{0b11..11}) = \texttt{0b11..11},$$

where `0b11..11` is the binary representation of $2^{s_i} - 1 \in \mathbb{Z}_{2^{s_i}}$. Moreover, if $p$ is of the form $p_{\texttt{gen1}}$, we also require that `0b00..00` $\in \mathbb{F}_2^{s_i}$ is a fixed point, i.e.,

$$\mathcal{S}_i(\texttt{0b00..00}) = \texttt{0b00..00}.$$

We define each permutation $\mathcal{S}_i$ as an invertible shift-invariant function defined via a local map $\Omega_i : \mathbb{F}_2^{l_i} \to \mathbb{F}_2$ for a certain $3 \leq l_i \leq s_i$, i.e.,

$$\mathcal{S}_i(z_1, \ldots, z_{s_i}) = \Omega_i(z_1, \ldots, z_{l_i}) \,||\, \Omega_i(z_2, z_3, \ldots, z_{l_i+1}) \,||\, \cdots \,||\, \Omega_i(z_{s_i}, z_1, \ldots, z_{l_i-1}), \tag{6}$$

where $z_1, z_2, \ldots, z_{s_i} \in \mathbb{F}_2$ and where the sub-indices are computed modulo $s_i$. For the local map, we use one proposed in Daemen's PhD thesis [Dae95, Table A.1], so that each $\mathcal{S}_i$ is invertible and satisfies the previous requirement. In particular, let $\oplus$ denote the `XOR` operation and $\odot$ the `AND` operation, respectively. If $\gcd(s_i, 2) = 1$, we suggest to use the chi-function $\chi : \mathbb{F}_2^3 \to \mathbb{F}_2$ (that instantiates Keccak/SHA-3) defined as

$$\chi(x_1, x_2, x_3) = x_1 \oplus (x_2 \oplus 1) \odot x_3.$$

If $\gcd(s_i, 2) \neq 1$ and $\gcd(s_i, 3) = 1$, we suggest to use $\psi : \mathbb{F}_2^4 \to \mathbb{F}_2$ defined as

$$\psi(x_1, x_2, x_3, x_4) = x_1 \oplus (x_2 \oplus 1) \odot x_3 \odot x_4.$$

If necessary, we suggest to apply a rotation of the bits at the output of $\mathcal{S}_i$ in order to reduce the number of fixed points.

### 4.3 Composition $\mathcal{C}$

The final operation of `Bar` is the inverse of the decomposition. Given $(x_1, x_2, \ldots, x_m) \in \mathbb{F}_2^{s_1} \times \mathbb{F}_2^{s_2} \times \cdots \times \mathbb{F}_2^{s_m}$, it yields an element $y \in \mathbb{F}_p$. For this purpose, first each element $x_i \in \mathbb{F}_2^{s_i} \equiv \mathbb{Z}_2^{s_i}$ is mapped into $x'_i \in \mathbb{Z}_{2^{s_i}}$.[5] Then, if $p = p_{\texttt{gen1}}$, $y$ is defined as

$$y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} \cdot x'_{\pi(i)} + 2^{\eta} \cdot \left( \sum_{i=m+1}^{n} 2^{\sum_{j=1}^{i} s_j} \cdot x'_{\Pi(i)} \right),$$

where $\pi$ is a permutation of $\{1, 2, \ldots, l\}$ and $\Pi$ is a permutation of $\{l+1, l+2, \ldots, m\}$. If $p = p_{\texttt{gen2}}$, $y$ is defined as

$$y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} \cdot x'_{\pi(i)},$$

where $\pi$ is a permutation of $\{1, 2, \ldots, m\}$.

### 4.4 Well-Definition and Bijectivity

Here we prove that our `Bars` layer and in particular its `Bar` components are invertible and well-defined. For simplicity, we omit the final rotation if existing and note that this operation is naturally invertible and maps `0b00..00` to `0b00..00` and `0b11..11` to `0b11..11`.

**Lemma 1.** *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(0) = 0$ and $\mathcal{S}_i(2^{s_i} - 1) = 2^{s_i} - 1$, where $i \in \{1, 2, \ldots, m\}$. If $p = 2^n - 1$, `Bar` maps elements from $\mathbb{F}_p$ to elements from $\mathbb{F}_p$.*

*Proof.* Let

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i, \quad y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i$$

be the decomposition of an input $x \in \mathbb{F}_p$ and the corresponding output $y$, respectively, where $y'_i = \mathcal{S}_i(x'_i)$ and $s_1 + s_2 + \cdots + s_m = n$. By definition, the application of all $\mathcal{S}_i$ does not extend $y$ to more than $\lceil \log_2(p) \rceil = n$ bits. Further, the output $2^n - 1$ can never be reached, since by definition $x < 2^n - 1$ and $x'_i \neq 2^{s_i} - 1 \implies \mathcal{S}_i(x'_i) = y'_i \neq 2^{s_i} - 1$ (recall that $2^{s_i} - 1$ is a fixed point for $\mathcal{S}_i$). It follows that $y < 2^n - 1$ and hence $y \in \mathbb{F}_p$. □

**Lemma 2.** *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(0) = 0$ and $\mathcal{S}_i(2^{s_i} - 1) = 2^{s_i} - 1$, where $i \in \{1, 2, \ldots, m\}$. If $p = 2^n - 2^\eta + 1$, `Bar` maps elements from $\mathbb{F}_p$ to elements from $\mathbb{F}_p$.*

*Proof.* Let

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i, \quad y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i$$

be the decomposition of an input $x \in \mathbb{F}_p$ and the corresponding output $y$, respectively, where $y'_i = \mathcal{S}_i(x'_i)$ and $s_1 + s_2 + \cdots + s_j = \eta, s_{j+1} + s_{j+2} + \cdots + s_m + \eta = n$ for some $j \in \{1, 2, \ldots, m\}$. By definition, the application of all $\mathcal{S}_i$ does not extend $y$ to more than $\lceil \log_2(p) \rceil = n$ bits. In $\mathbb{F}_p$, $p - 1$ (i.e., the largest possible input element) is given by $2^\eta \cdot (2^{n-\eta} - 1)$, and in this case $(x'_1, x'_2, \ldots, x'_j) = (0, 0, \ldots, 0)$ and $(x'_{j+1}, x'_{j+2}, \ldots, x'_m) = (1, 1, \ldots, 1)$. Since $\mathcal{S}_i(0) = 0$ for $i \in \{1, 2, \ldots, j\}$ and $\mathcal{S}_i(2^{s_i} - 1) = 2^{s_i} - 1$ for $i \in \{j+1, j+2, \ldots, m\}$, $p - 1$ is a fixed point under the application of `Bar`. For any other element in $\mathbb{F}_p$, either $(x'_1, x'_2, \ldots, x'_j) = (0, 0, \ldots, 0)$ or, if this is not the case, $(x'_{j+1}, x'_{j+2}, \ldots, x'_m) \neq (1, 1, \ldots, 1)$. Together with the fact that $2^\eta \cdot (2^{n-\eta} - 1)$ is a fixed point when applying $\mathcal{S}_i$ for $i \in \{j+1, j+2, \ldots, m\}$, it follows that $y < p$ and hence $y \in \mathbb{F}_p$. □

**Lemma 3.** *If $p \in \{p_{\texttt{gen1}} = 2^n - 2^\eta + 1, p_{\texttt{gen2}} = 2^n - 1\}$, the `Bar` operation*

$$\texttt{Bar} = \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x)$$

*is invertible.*

[5] Note that in this case $x_i$ has already been transformed by $\mathcal{S}_i$.

*Proof.* Let

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i, \quad y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i$$

be the decomposition of an input $x \in \mathbb{F}_p$ and the corresponding output $y$, respectively, where $y'_i = \mathcal{S}_i(x'_i)$. First, note that $\mathbb{F}_2^n$, $\mathbb{F}_{2^n}$, and $\mathbb{Z}_{2^n}$ are isomorphic to each other. Then the operations $\mathcal{C}$ and $\mathcal{D}$ are invertible by definition, since they only consist of different representations of $x$ and $y$. Finally, each $\mathcal{S}_i$ is invertible by definition, from which it follows that $y'_i = \mathcal{S}_i(x'_i) \implies x'_i = \mathcal{S}_i^{-1}(y'_i)$ for a well-defined function $\mathcal{S}_i^{-1}$. □

## 5 Security Analysis

In this section, we propose a security analysis of our design. To summarize, we are not able to break 6 rounds of the proposed scheme or a weaker version of it (i.e., without some of the components) with any basic attacks proposed in the literature. As future work, we encourage to study reduced-round or/and toy variants of our design.

### 5.1 Differential Cryptanalysis

Given pairs of inputs with some fixed input differences, differential cryptanalysis [BS90] considers the probability distribution of the corresponding output differences produced by the cryptographic primitive. Let $\Delta_I, \Delta_O \in \mathbb{F}_p^t$ be respectively the input and the output differences through a permutation $\mathcal{P}$ over $\mathbb{F}_p^t$. The differential probability (DP) of having a certain output difference $\Delta_O$ given a particular input difference $\Delta_I$ is equal to

$$\mathrm{Prob}_{\mathcal{P}}(\Delta_I \to \Delta_O) = \frac{|\{x \in \mathbb{F}_p^t \mid \mathcal{P}(x + \Delta_I) - \mathcal{P}(x) = \Delta_O\}|}{p^t}.$$

In the case of iterated schemes, a cryptanalyst searches for ordered sequences of differences over any number of rounds that are called differential characteristics/trails. Assuming the independence of the rounds, the DP of a differential trail is the product of the DPs of its one-round differences.

As is well-known, the maximum differential probability of the square map is $1/p$. Here we compute the minimum number of active square maps over $r$ rounds. Since the `Bars` layer is not supposed to have good statistical properties, we simply assume that the attacker can skip it with probability 1. Hence, we omit it in our analysis.

Denote the number of active words in the input and the output of the $i$-th `Bricks` layer by $a_i$ and $b_i$, respectively. Then we exploit two properties:

- Each active input word (different from the last one) activates at least 1 squaring in `Bricks`. Hence, $a \geq 1$ words activate at least $a - 1 \geq 0$ squarings.
- Each active output word (different from the last one) implies that 1 squaring is active, for this word or the left one. Hence, $b \geq 1$ words activate $\frac{b-1}{2} \geq 0$ squarings.

Together with the MDS property, implying $b_i + a_{i+1} \geq t+1$ for each $i \geq 1$, we obtain the following inequalities for the number of active squarings $s_i$:

$$\begin{aligned}
s_1 &\geq \max\left\{(a_1 - 1, \frac{b_1 - 1}{2}\right\}, \quad b_1 + a_2 \geq t + 1,\\
s_2 &\geq \max\left\{a_2 - 1, \frac{b_2 - 1}{2}\right\}, \quad b_2 + a_3 \geq t + 1,\\
s_3 &\geq \max\left\{a_3 - 1, \frac{b_3 - 1}{2}\right\}, \quad b_3 + a_4 \geq t + 1,\\
&\vdots\\
s_{r-1} &\geq \max\left\{a_{r-1} - 1, \frac{b_{r-1} - 1}{2}\right\}, \quad b_{r-1} + a_r \geq t + 1,\\
s_r &\geq \max\left\{a_r - 1, \frac{b_r - 1}{2}\right\},
\end{aligned}$$

where $r$ is the number of rounds. We propose the following lemma for finding the minimum number of active square maps.

**Lemma 4.** *A set of real positive values $(s_1, s_2, \ldots, s_r)$, which minimizes the expression $s_{max} := s_1 + s_2 + s_3 + \ldots + s_r$ and $s_i \geq 0$, $a_i \geq 1$, $b_i \geq 1$, satisfies*

$$s_r = 0 \qquad \textit{and} \qquad s_{r-i} = \frac{t-1}{3} \cdot \left(1 + \frac{(-1)^{i+1}}{2^i}\right) \textit{ for each } i > 0. \tag{7}$$

*Proof.* Consider an optimal tuple $(s_1, s_2, \ldots, s_r)$. Note first that such a tuple turns all inequalities into strict equations, as otherwise we can reduce $s_{\max}$. Now consider any MDS property $b_i + a_{i+1} = t+1$. If $(b_i - 1)/2 < s_i$, we can increase $b_i$ to make those equal and not to increase $s_{\max}$. Similarly, if $a_{i+1} < s_{i+1}$, we can increase $a_i$ to make those equal and not to increase $s_{\max}$. Thus we conclude that for an optimal tuple the values $b_i$ and $a_{i+1}$ are the maximums that determine $s_i$ and $s_{i+1}$ respectively. This simplifies our system, i.e.,

$$\begin{aligned}
s_1 &= \frac{b_1 - 1}{2}, \quad b_1 + a_2 = t + 1, \\
s_2 = a_2 - 1 &= \frac{b_2 - 1}{2}, \quad b_2 + a_3 = t + 1, \\
s_3 = a_3 - 1 &= \frac{b_3 - 1}{2}, \quad b_3 + a_4 = t + 1, \\
&\vdots \\
s_r &= a_r - 1,
\end{aligned}$$

and even further, i.e.,

$$\begin{aligned}
2s_1 + s_2 &= t - 1, \\
2s_2 + s_3 &= t - 1, \\
2s_3 + s_4 &= t - 1, \\
&\vdots \\
2s_{r-1} + s_r &= t - 1, \\
s_r &= a_r - 1.
\end{aligned}$$

It is simple to note that if $s_r > 0$, then we could decrease $s_{\max}$. Indeed, if we decrease $s_r$ to 0, we would have to increase $s_{r-1}$ by $s_r/2$, then decrease $s_{r-2}$ by $s_r/4$ and so on, altogether decreasing $s_{\max}$ by $s_r \cdot (1 - 1/2 + 1/4 - 1/8 + \cdots)$, where $(1 - 1/2 + 1/4 - 1/8 + \cdots) > 0$. Note also that for $s_r \leq t - 1$ all other $s_i$ are non-negative. Thus, the minimum is achieved by $s_r = 0$ and

$$s_{r-1} = \frac{t-1}{2}, \quad s_{r-2} = \frac{t-1}{4}, \quad s_{r-3} = \frac{3(t-1)}{8}, \quad \ldots, \quad s_{r-i} = \frac{t-1}{3} \cdot \left(1 + \frac{(-1)^{i+1}}{2^i}\right).$$

This completes the proof. □

Thus we have the following bounds for the total number of active squarings.

- 2 rounds: $s_{\max} \geq \frac{t-1}{2}$;
- 3 rounds: $s_{\max} \geq \frac{3(t-1)}{4}$;
- 4 rounds: $s_{\max} \geq \frac{9(t-1)}{8}$;
- 5 rounds: $s_{\max} \geq \frac{23(t-1)}{16}$;

Since the maximum differential probability of a squaring is $1/p$, we get the following.

**Corollary 1.** *Any 4-round differential characteristic for* `Monolith` *has a probability of at most* $p^{\frac{-9(t-1)}{8}}$.

As a result, any characteristic that spans over 5 rounds and more would cover more squarings than the number of state elements, and thus a solution to it cannot be found by standard means. Therefore, a differential-based collision attack on 5 rounds looks infeasible.

### 5.2 Other Statistical Attacks

We claim that 6 rounds are sufficient for preventing other statistical attacks as well. Here we provide argument to support such conclusion for one of the most powerful statistical attacks against a hash function, that is, the rebound attack. For that goal, we propose an analysis of the number of the fixed points and of the truncated differential characterestics.

**Fixed Points.** Contrary to `Reinforced Concrete`, the `Bars` layer of `Monolith` has very few fixed points, thanks to the prime number form and an extra bit shift.

Both local maps $x \oplus \big((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3)\big)$ and $x \oplus \big((\overline{x} \lll 1) \odot (x \lll 2)\big)$ have about $(7/4)^n$ fixed points (for even and odd $n$, respectively) when considered over $\mathbb{F}_2^n$ (a bit value is preserved if the product of nearby bits is 0). However, all of them except $\mathbf{0}$ and $\mathbf{1} = 2^n - 1$ are destroyed by the circular shift (verified experimentally).

A `Bar` of `Monolith`-64, consisting of 8 such S-boxes, admits $2^8 - 2^4 + 1 = 241$ fixed points out of $2^{64} - 2^{32} + 1$. This implies that the probability that a point is fixed is approximately $2^{-56}$ for `Bar` and less than $2^{-56 \cdot 4} = 2^{-224}$ for `Bars`. Similarly, a `Bar` of `Monolith`-31 admits $2^4 - 1 = 15$ fixed points out of $2^{31} - 1$. This implies that the probability that a point is fixed is approximately $2^{-27}$ for `Bar` and less than $2^{-27 \cdot 8} = 2^{-216}$ for `Bars`.

For comparison, we recall that a `Bar` of `Reinforced Concrete` has $2^{134.5}$ fixed points out of $2^{254}$ possibilities. Hence, the probability of encountering a fixed point is approximately $2^{-119.5 \cdot 3} = 2^{-358.5}$ for `Bars`. At the current state of the art, we are not aware of any attack that exploits these fixed points.

**Truncated Differential and Rebound Attacks.** Truncated differential attacks [Knu94] are used mostly against primitives that have incomplete diffusion over a few rounds. This is not the case here as the `Concrete` matrix is MDS. We have not found any other attacks where a truncated differential can be used as a subroutine either.

Rebound attacks [MRS+09] are widely used to analyze the security of various types of hash functions against shortcut collision attacks since the beginning of the SHA-3 competition. It starts by choosing internal state values in the middle of the computation, and then computing in the forward and backward directions to arrive at the inputs and outputs. It is useful to think of it as having central (often called "inbound") and the above mentioned "outbound" parts. In the attack, solutions to the inbound phase are first found, and then are filtered in the outbound phase.

Whereas it is not possible to prove the resistance to the rebound attacks rigorously, we can provide some meaningful arguments to demonstrate that they are not feasible. The inbound phase deals with truncated and regular differentials. By Corollary 1 we see that a solution for a 5-round differential cannot be found, and so the inbound phase cannot cover more than 4 `Bricks` layers. In the outbound phase, the `Concrete` layers that surround these `Bricks` layers make all differentials diffuse to the entire state, so that the next `Bricks` layers destroy all of those. We hence conclude that 6 rounds of `Monolith` are sufficient to prevent rebound attacks.

The best attack of this kind that we were able to conduct ourselves is a near-collision attack on the reduced 3-round permutation without the `Bars` layer. In our attack we show how to find a state that satisfies a differential $\Delta_1 \to \Delta_8$ for certain $\Delta_1, \Delta_8$ which are equal in the last $\mathbb{F}_p$ word, i.e., $\Delta_{1,t} = \Delta_{8,t}$. As a concrete application, this yields a zero difference in this word for the compression function $x \mapsto \mathrm{Trunc}_n(\mathcal{P}(x) + x)$, which is a near-collision.

The inbound phase covers 3 layers of `Bricks` separated by 2 `Concrete` layers:

$$\Delta_1 \xleftarrow[t \to 1]{\texttt{Concrete}} \Delta_2 \xleftarrow[1]{\texttt{Bricks}} \underbrace{\Delta_3 \xrightarrow[1 \to t]{\texttt{Concrete}} \Delta_4 \xleftarrow[t]{\texttt{Bricks}} \Delta_5 \xrightarrow[t \leftarrow 2]{\texttt{Concrete}} \Delta_6 \xrightarrow[2]{\texttt{Bricks}} \Delta_7}_{\text{inbound phase}} \xrightarrow[2 \to t]{\texttt{Concrete}} \Delta_8.$$

To find such a state pair, we apply the following approach.

1. In the inbound phase we arbitrarily choose $\delta$ and set $\Delta_3 = [0, 0, \ldots, 0, \delta]$ such that its non-zero difference is in the last word only and propagates through $\texttt{Bricks}^{-1}$ untouched. That is, $\Delta_2 = \Delta_3$. Let $\Delta_1$ be $\texttt{Concrete}^{-1}(\Delta_2)$.

2. The inbound phase covers the expansion of $\Delta_2$ to $t$ words and back to the 2-word difference $\Delta_7 = [0, 0, \ldots, 0, \delta_2, \delta_3]$. Note that we have $\Delta_6 = [0, 0, \ldots, 0, \delta_2, \delta_4]$. We arbitrarily set $\delta_2, \delta_3$ such that $\Delta_{8,t} = \Delta_{1,t}$ and then choose $\delta_4$ such that

$$\texttt{Concrete}(\Delta_2) = \Delta_{4,1} = \Delta_{5,1} = \texttt{Concrete}^{-1}(\Delta_6).$$

3. As a result, the differential path for the full 3-round scheme is established, and we determine the state. The $(\delta_3, \delta_4)$ differential determines the input word $x_{t-1}$ of the third `Bricks` layer, and the equation

$$\texttt{Bricks}(\mathbf{X} + \Delta_4) = \texttt{Bricks}(\mathbf{X}) + \Delta_5.$$

determines input words $x_1, x_2, \ldots, x_{t-1}$ of the second `Bricks` layer. Note that this is a system of linear equations, and solving it we can determine the full state.

Overall we obtain a partial collision at a negligible cost (the cost for solving the linear system of equations can be approximated by $\mathcal{O}(t^3)$, which is much smaller than the cost for constructing the collision in the case of a random permutation approximated by $\mathcal{O}(p^{1/2})$). We are not aware of any possible extension of such attack to more rounds and/or including `Bars`, which is left as an open problem for future work.

### 5.3 Algebraic Analysis: Degree and Density of the `Bars` Polynomials

**Generic Lemmas.** We first establish several lemmas that are valid for all primes.

**Lemma 5.** *Let $p \geq 3$ be a prime number, and let $\mathcal{F}$ denote the squaring function $x \to x^2$ over $\mathbb{F}_p$. Let $\mathcal{F}'$ be any interpolant of $\mathcal{F}$ over $\mathbb{F}_2^{\lceil \log_2 p \rceil}$, i.e. for any $a < p$ and its bit representation $a'$ we have that $\mathcal{F}'(a')$ is the bit representation of $\mathcal{F}(a)$. Then $\mathcal{F}'$ has degree at least $d$, where $d$ is the maximum positive integer such that $d < \log_2 \sqrt{p}$ and $\lceil 2^{d-0.5} \rceil$ is odd.*

*Proof.* We prove this result by contradiction. Suppose that the degree of $\mathcal{F}'$ is smaller than $d$. Then the `XOR` sum of its output over any hypercube of degree $d$ is equal to zero [Lai94], including the hypercube

$$\mathfrak{H} := \{a_0 = (0, 0, \ldots, 0), \ldots, a_{2^d-1} = (0, \ldots, 0, \underbrace{1, \ldots, 1}_{d \text{ ones}})\}.$$

Note that $\mathcal{F}(a_i) = i^2 < p$ by the definition of $d$. Now consider $\mathfrak{B} = \{a_i \in \mathfrak{H} \mid i > 2^{d-0.5}\}$, so that (i) $2^{2d} > \mathcal{F}(b \in \mathfrak{B}) > 2^{2d-1}$, and (ii) the $2d$-th least significant bit is set. By simple computation, the size of $\mathfrak{B}$ is $2^d - \lceil 2^{d-0.5} \rceil$. Whenever this number is odd, $\mathcal{F}$ does not `XOR` to 0 at the $2d$-th least significant bit, which contradicts the previous fact. As a result, the squaring has at least degree $d$ if $d \in \mathfrak{D}$ and $d < \log_2 \sqrt{p}$. □

For example, $\lceil 2^{d-0.5} \rceil$ is odd for $d \in \mathfrak{D} = \{2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 21, 22, 25, 26, 29, 30, \ldots\}$.

**Lemma 6 (Differential).** *Let $F$ be a function that maps $\mathbb{F}_p$ to itself with a differential $\Delta_I \to \Delta_O$ holding with probability $0 < \alpha < 1$, that is, $\frac{|\{x \in \mathbb{F}_p | \mathcal{F}(x + \Delta_I) = \mathcal{F}(x) + \Delta_O\}|}{p} = \alpha$. Then we have*

$$\deg(\mathcal{F}) > \alpha \cdot p, \tag{8}$$

*where $\deg(\mathcal{F})$ is the degree of $\mathcal{F}$ as a polynomial over $\mathbb{F}_p$.*

*Proof.* By definition, the equation $\mathcal{F}(x + \delta_{in}) = \mathcal{F}(x) + \delta_{out}$ has at most $\alpha \cdot p < p$ solutions $x_1, x_2, \ldots, x_{\alpha p}$. Therefore, the polynomial $\mathcal{G}(x) := \mathcal{F}(x + \delta_{in}) - \mathcal{F}(x) - \delta_{out}$ is divisible by the polynomial $(x - x_1) \cdot (x - x_2) \cdot \cdots \cdot (x - x_{\alpha \cdot p})$ of degree $\alpha \cdot p$, and so it has a degree of at least $\alpha \cdot p$. As the degree of the polynomial $\mathcal{G}$ is smaller than the degree of $\mathcal{F}$ by 1, we obtain that $\deg(\mathcal{F}) > \alpha \cdot p$. □

**Lemma 7 (Linear Approximation).** *Let $F$ be a function that maps $\mathbb{F}_p$ to itself such that there exists a linear approximation $(a, b)$ with probability $0 < \beta < 1$, that is, $\frac{|\{x \in \mathbb{F}_p | \mathcal{F}(x) = a \cdot x + b\}|}{p} = \beta$. Then we have*

$$\deg(\mathcal{F}) \geq \beta \cdot p\,. \tag{9}$$

*Proof.* By definition, the equation $\mathcal{F}(x) = A \cdot x + B$ has at most $\beta \cdot p$ solutions $x_1, x_2, \ldots, x_{\beta \cdot p}$. Therefore, the polynomial $\mathcal{G}(x) := \mathcal{F}(x) - (A \cdot x + B)$ is divisible by the polynomial $(x - x_1) \cdot (x - x_2) \cdot \cdots \cdot (x - x_{\beta p})$ of degree $\beta \cdot p$. Similar to before, we can conclude that $F$ has degree at least equal to $\beta \cdot p$. □

Based on the previous result, we can immediately conclude the following.

**Corollary 2.** *Let $\mathcal{F}$ be a function that maps $\mathbb{F}_p$ to itself with $b < p$ fixed points, that is, $|\{x \in \mathbb{F}_p : F(x) = x\}| = b$. It follows that*

$$\deg(\mathcal{F}) \geq b\,. \tag{10}$$

**Lower Bound on the Degree over $\mathbb{F}_2$.**

**Proposition 1.** *Let $p \in \{p_{\texttt{Mersenne}}, p_{\texttt{Goldilocks}}\}$. Let $\mathcal{F}'$ be an interpolant over $\mathbb{F}_2^{\lceil \log_2 p \rceil}$ of the squaring operation $\mathcal{F}(x) = x^2$ over $\mathbb{F}_p$. Then $\mathcal{F}'$ has degree at least $d$, where*

- *$d = 30$ for $p = 2^{64} - 2^{32} + 1$,*
- *$d = 15$ for $p = 2^{31} - 1$.*

As the squaring operation is a component of `Bricks`, we get that it has degree $d \geq 30$ as well.

**Lower Bound on the Degree over $\mathbb{F}_p$.**

**Lemma 8.** *Let $n > 4$.*

- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (4)*

$$y \mapsto \left(y \oplus \left((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\right)\right) \lll 1$$

  *is at least 13/32.*
- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (5)*

$$y' \mapsto \left(y' \oplus \left((\overline{y'} \lll 1) \odot (y' \lll 2)\right)\right) \lll 1$$

  *is at least 1/8.*

*In particular we have input pairs of form $(x_1, x_2, \ldots, x_{n-1}, 0), (x_1, x_2, \ldots, x_{n-1}, 1)$ mapping to $(y_1, y_2, \ldots, 0, y_n), (y_1, y_2, \ldots, 1, y_n)$ with at least the same probability (13/32 and 1/8, resp.).*

*Proof.* Consider two input states $x, y$ with a single bit difference in bit $i$ such that $x_i = 1 - y_i = 0$. Let us derive sufficient conditions when the output states $x', y'$ differ in bit $i - 1$ only and $x'_i = 1 - y'_i = 0$. This happens when the product in the S-box bit mapping is 0 whenever bit $i$ is `XOR`ed or is part of the product, i.e.,

$$\begin{aligned} \overline{y_{i+1}} \odot y_{i+2} \odot y_{i+3} &= 0, \\ y_{i+1} \odot y_{i+2} &= 0, \\ \overline{y_{i-1}} \odot y_{i+1} &= 0, \\ \overline{y_{i-2}} \odot y_{i-1} &= 0. \end{aligned}$$

The number of 5-tuples satisfying this system is 13 out of 32 possible. Therefore, a differential holds with probability 13/32.

For the S-box Eq. (5) we observe that a single bit difference in bit $i$ is mapped to a single bit difference in bit $i - 1$ if

$$\begin{aligned} \overline{y_{i+1}} \odot y_{i+2} &= 0, \\ y_{i+1} &= 0, \\ \overline{y_{i-1}} &= 0, \end{aligned}$$

which holds for one 3-tuple out of 8 ones. Therefore, the differential holds with probability 1/8. □

**Table 2.** Degree and density of the polynomials resulting from `Bar` applied to various field elements.

| $p$ | Bit splittings | Degree | Density |
|---|---|---|---|
| $2^8 - 2^4 + 1$ | $\{4, 4\}$ | 239 $(= p - 2)$ | 100% |
| $2^{13} - 2^8 + 1$ | $\{8, 5\}, \{4, 4, 5\}$ | 7935 $(= p - 2)$ | $> 99\%$ (7934/7935) |
| $2^{13} - 2^5 + 1$ | $\{5, 8\}, \{5, 4, 4\}$ | 8159 $(= p - 2)$ | $> 99\%$ (8157/8159) |
| $2^{14} - 2^{10} + 1$ | $\{10, 4\}, \{5, 5, 4\}$ | 15359 $(= p - 2)$ | $> 99\%$ (15358/15359) |
| $2^{14} - 2^4 + 1$ | $\{4, 10\}$ | 16367 $(= p - 2)$ | 100% |
| $2^{14} - 2^4 + 1$ | $\{4, 5, 5\}$ | 16367 $(= p - 2)$ | $> 99\%$ (16364/16367) |
| $2^{13} - 1$ | $\{5, 8\}, \{8, 5\}, \{4, 9\}, \{9, 4\}$ | 8189 $(= p - 2)$ | $> 99\%$ (8188/8189) |
| $2^7 - 1$ | $\{3, 4\}, \{4, 3\}$ | 125 $(= p - 2)$ | $> 99\%$ (124/125) |
| $2^5 - 1$ | – | 26 $(= p - 5)$ | $\approx 21\%$ (6/29) |
| $2^7 - 1$ | – | 120 $(= p - 7)$ | $\approx 14\%$ (18/125) |
| $2^{13} - 1$ | – | 8178 $(= p - 13)$ | $\approx 8\%$ (629/8189) |

**Lemma 9.** *The* `Bar` *function for* $p = 2^{64} - 2^{32} + 1$. *The* `Bar` *function for* $p = 2^{31} - 1$ *has a differential probability of at least* $2^{-1.2}$.

*Proof.* The differential probability of `Bar` as a function over $\mathbb{F}_2$ is at least the probability of a single S-box, as we can select inputs that activate only one S-box. By Lemma 8 the $\mathbb{F}_2$-differential in the last bit implies the $\mathbb{F}_p$ differential $1 \to 2$ of the same probability. When 8 S-boxes are used, the $\mathbb{F}_2^{64}$ differential holds for at least $13 \cdot 2^{59}$ 64-bit inputs. To get to $\mathbb{F}_p$ we should exclude from those the ones that possibly exceed $p$, i.e., $2^{32}$ ones. The probability is then lower-bounded by $2^{-1.4}$.

Similarly, for 31-bit inputs, Lemma 8 implies that $3 + 1$ concatenated S-boxes together yield a differential probability of at least 13/32 (we activate the weaker 8-bit S-box) both when viewed over $\mathbb{F}_2^{31}$ and over $\mathbb{F}_p$. □

**Proposition 2.** *The* `Bars` *operation (and its inverse) has degree at least*

- $2^{62}$ *for* $p = 2^{64} - 2^{32} + 1$;
- $2^{29}$ *for* $p = 2^{31} - 1$.

**Degree and Density over $\mathbb{F}_p$: Practical Results.** Evaluating the actual density of the polynomial resulting from `Bar` applied to a single field element in $\mathbb{F}_p$, where $p \in \{2^{64} - 2^{32} + 1, 2^{31} - 1\}$, is infeasible in practice. Indeed, any enumeration and subsequent interpolation approach would take far too long.

Therefore, in our experiments we focus on smaller finite fields defined by "similar" prime numbers. In particular, we focus on $n$-bit primes of the form $2^n - 2^\eta + 1$ for $\eta$ as close to $n$ as possible. We then apply the S-box $S_i$ to smaller parts of the field element, exactly as in `Bar` where the S-box is applied to each 8-bit part of the larger field element. We also vary the sizes of the parts to which the $S_i$ are applied in order to get a broader picture.

The results of our evaluation are shown in Table 2. For example, in the first case, where $p = 2^8 - 2^4 + 1$, $S_i$ is applied to the first 4 bits (starting from the least significant bit) and then to the next 4 bits, covering the entire field element. The size of these parts is indicated in the second column. As we can see, the maximum degree is reached for all tested primes of the form $2^n - 2^\eta + 1$, where $\eta > 1$. Moreover, for these primes, the density is always close to 100%, mostly matching it. We also applied $S_i$ to elements of $\mathbb{F}_{2^n - 1}$ directly, where $n \in \{5, 7, 13\}$, which resulted in almost maximum-degree polynomials of low density (specifically, only 6, 18, and 630 monomials exist in the polynomial representation, respectively). This suggests that increasing the number of S-box applications per field element (i.e., increasing the number of smaller parts to which $S_i$ are applied) is beneficial for the density of the resulting polynomial.

We also evaluated the degrees and density values resulting from the inverse S-boxes applied to the field elements, in order to get an estimation of the algebraic strength of the inverse operation. The results match the results given in Table 2, where always more than 99% monomials are reached together with a degree close to the maximum.

**Table 3.** Degree and density of the polynomials after a single round, where $t = 4$ and two input variables are used (with the other two input elements being fixed).

| $p$ | Bit splittings | Degree | Density |
|---|---|---|---|
| $2^8 - 2^4 + 1$ | $\{4, 4\}$ | 239 $(= p - 2)$ | $> 99\%$ (28785/28920) |
| $2^7 - 1$ | $\{3, 4\}$ | 125 $(= p - 2)$ | $> 98\%$ (7919/8001) |
| $2^7 - 1$ | $\{4, 3\}$ | 125 $(= p - 2)$ | $> 98\%$ (7919/8001) |

**Degree and Density over $\mathbb{F}_p^t$: Practical Results.** We also ran tests regarding the density over the entire state. Naturally, this task gets harder with an increased number of rounds, since the degrees are rising too quickly. In our tests we focused on $p \in \{2^8 - 2^4 + 1, 2^7 - 1\}$ and $t = 4$, and we give the results together with the sizes of the smaller S-boxes in Table 3.

As can be seen, the maximum number of monomials is almost reached after a single round. We suspect that some of the monomials are not reached due to cancellations, which is reasonable when considering these small prime fields. Still, we acknowledge this fact by adding another round on top of that in order to ensure that all polynomial representations of the state are dense and of maximum degree. Thus, having 6 rounds achieves 4 rounds of security margin regarding degrees and density of polynomials.

### 5.4 The CICO Problem for Keyless Algebraic Attacks

In recent circuit-friendly hash functions, the CICO problem described in the following has often been exploited in order to argue security against some classes of algebraic attacks (in particular Gröbner basis ones).

**Definition 1 (CICO Problem).** *A permutation $\mathcal{P} : \mathbb{F}_p^t \to \mathbb{F}_p^t$ provides $\kappa$ bits of security against the $v$-CICO problem if no algorithm with expected complexity smaller than $p^v$ finds $I_1 \in \mathbb{F}_p^{t-v}$ and $O_2 \in \mathbb{F}_p^{t-v}$ such that $\mathcal{P}(I_1 \,||\, \underbrace{\mathbf{0}}_{v \text{ words}}) = \underbrace{\mathbf{0}}_{v \text{ words}} \,||\, O_2$.*

The relation between the CICO problem and the preimage security of a hash function is for example described in [GHR+22]. In particular, solving it yields a preimage solution for a sponge hash function with $v$ capacity elements and $v$ output elements, and hence solving this instance of the problem must not require fewer than $p^{v/2}$ operations.

To express the CICO problem algebraically, we first interpret the output elements as polynomials of the input elements. Then, we find a solution to the system of $v$ polynomial equations of $t - v$ input variables (as the remaining $v$ ones are set to zero). Let us now consider two ways of solving this system.

*Remark 1.* For completeness, we point out that the strategies used for solving the CICO problem can be also adapted to attack the case in which our design is used for authenticated encryption. In such a case, the attacker observes several outputs of a permutation where part of the input is secret, that is,

$$\forall i \in \{1, 2, \dots\} : \quad \mathcal{P}(k_i \,||\, x_i) = c_i \,||\, y_i,$$

where $x_i$ and $y_i$ are available to the attacker, while $c_i$ (the inner part) and $k$ are secret. Hence, we remark that the security against CICO implies the security against this problem as well.

**Univariate Case.** One way to solve a multivariate system is to make it univariate by guessing $t - v - 1$ variables. Note that our guess may be invalid if the number of equations exceed the number of variables so we have to repeat the guess $p^{v-1}$ times.

- If $v = 1$, we have to solve a single polynomial equation faster than in time $p$. The degree of the polynomial reaches $p$ after 2 applications of the `Bars` layer, i.e., after 2 rounds. Therefore, solving the equation will require time $\approx p$.
- If $v > 1$, we have several polynomial equations of degree close to $p$. Solving a system of univariate dense polynomials of degree $d$ is close to $d$, so we expect spending at least time $p$ to verify the guess. Therefore the total complexity still exceeds $p^v$.

**Multivariate Case: Gröbner Bases.** In a more general case we work with a system of $v$ polynomial equations of $t - v$ input variables. The system likely remains solvable if we guess extra $t-2v$ variables to have both $v$ equations and variables. The main technique of solving these systems is to use Gröbner bases, as described with the following steps.

1. Compute a Gröbner basis for the zero-dimensional ideal of the system of polynomial equations with respect to the *degrevlex* term order.
2. Convert the *degrevlex* Gröbner basis into a *lex* Gröbner basis using the FGLM algorithm [FGL+93].
3. Factor the univariate polynomial in the *lex* Gröbner basis and determine the solutions for the corresponding variable. Back-substitute those solutions, if needed, to determine solutions for the other variables.

The total complexity of a Gröbner basis attack is hence the sum of the respective complexities of the above steps. In our following analysis, we only estimate the complexity for computing a Gröbner basis in *degrevlex* order to argue security against this type of attack.

For a semi-regular input system $\mathcal{F}_1, \ldots, \mathcal{F}_k$ in $l$ variables with degrees $d_1, \ldots, d_k$, it is well-known that the Hilbert series of the ideal generated by $\mathcal{F}_1, \ldots, \mathcal{F}_k$ is related to the cost of computing a degrevlex-Gröbner basis, see [BFS+05]. The index of the first non-positive coefficient of the function

$$z \mapsto \frac{\prod_{i=1}^{k}(1 - z^{d_i})}{(1-z)^l}$$

is called degree of regularity $d_{\text{reg}}$ and it is used to establish the following upper bound for the complexity $C_{GB}$ (counting finite field operations) of computing a Gröbner basis in *degrevlex* order (via the matrix-F5 algorithm [Fau02]) of a semi-regular input system:

$$C_{\text{GB}}(l, d_{\text{reg}}) \in \mathcal{O}\left(\binom{l + d_{\text{reg}}}{l}^{\omega}\right), \tag{11}$$

where $\omega$ denotes the linear algebra constant. Most of the time, the algebraic model of a cryptographic primitive does *not* yield a semi-regular sequence, however, the case of a semi-regular sequence can be considered as the generic case. Colloquially speaking, 'generic case' in this context means a system of random equations. Thus, a comparison of the algebraic model with this generic case can still be an informative approach and help to establish heuristic bounds on the maximum degree in a Gröbner basis computation when practical experiments are no longer possible.

To estimate the complexity of computing a Gröbner basis in *degrevlex* order, a widely used heuristic approach is to compute such a basis for small-scale instances of the analysed primitive and to observe the maximum degrees reached during these computations. This degree can be used as in indicator of the final complexity. With this approach, an estimate for the maximum degree of the full-round primitive can be found by extrapolating the acquired data points.

**Algebraic Model for `Bar`.** We suggest the following algebraic model for `Bar` for a decomposition of a prime field element into $m$ buckets with sizes $2^{s_1}, 2^{s_2}, \ldots, 2^{s_m}$:

$$\begin{cases} x = x_1 b_1 + x_2 b_2 + \cdots + x_m b_m, \\ 0 = \prod_{i=j}^{2^{s_i}}(x_i - j), \quad 1 \le j \le m, \\ y = \mathcal{L}_1(x_1) b_1 + \mathcal{L}_2(x_2) b_2 + \cdots + \mathcal{L}_m(x_m) b_m. \end{cases}$$

Here, $b_1 = 1$ and $b_i := 2^{s_1 + \cdots + s_i}$ for $2 \le i \le m$ and $L_i(x_i)$ is the interpolation polynomial of degree $2^{s_i} - 1$ for S-box $\mathcal{S}_i$ given by

$$\mathcal{L}_i(x_i) := \sum_{1 \le k \le s_i} \mathcal{S}_i(k) \prod_{\substack{1 \le j \le s_i \\ j \ne k}} \frac{x_i - j}{k - j}.$$

The resulting system consists of $m + 2$ equations, namely $m$ equations of respective degrees $2^{s_1}, \ldots, 2^{s_m}$ and 2 equations of degree 1. The $m + 2$ variables are $x_1, \ldots, x_m, x, y$.

**Algebraic Model for One Round of `Monolith`.** We model one round of `Monolith` as a CICO problem with $t = 4$ words, i.e., we are looking for $x_2, x_3, x_4, y_2, y_3, y_4 \in \mathbb{F}_p$ such that

$$\mathcal{F}(0, x_2, x_3, x_4) = (0, y_2, y_3, y_4),$$

where $\mathcal{F} \coloneqq \texttt{Concrete} \circ \texttt{Bricks} \circ \texttt{Bars} \circ \texttt{Concrete}$ denotes a single round of `Monolith` with an added `Concrete` layer. For `Concrete`, we use the circulant matrix $M = \text{circ}(2, 1, 1, 1)$, which is not MDS and can thus be seen as an optimistic choice (from the attacker's perspective). We model the above CICO problem as a system of polynomial equations, which we solve using Gröbner basis techniques. To solve this problem we suggest an algebraic model such that

$$\begin{cases} 0 = \texttt{Concrete}^{-1}(u_1, u_2, u_3, u_4)_0, \\ v_1 = \texttt{Bar}(u_1), \\ v_2 = \texttt{Bar}(u_2), \\ 0 = (\texttt{Concrete} \circ \texttt{Bricks})(v_1, v_2, u_3, u_4)_0, \end{cases}$$

where $\mathcal{H}(\cdot)_i$ denotes the $i$-th element of the output of the function $\mathcal{H}$ for $i \in \{1, 2, 3, 4\}$. We note, each `Bar` function decomposes a prime field element into 2 buckets and $v_i = \texttt{Bar}(u_i)$ denotes above algebraic model for `Bar` with $m = 2$. The resulting equation system consists of 10 equations with

- 4 equations for each `Bar` system $v_i = \texttt{Bar}(u_i)$, $i = 1, 2$, and
- 2 equations for modelling the CICO constraint at the input and the output.

In total, we have 10 variables, namely $u_1, u_2, u_3, u_4, v_1, v_2$ and 2 internal variables for each `Bar` system. To estimate the cost of Gröbner basis computations, we use the well-known estimate

$$C_{\text{GB}} \in \mathcal{O}\left(\binom{n_v + d_{\max}}{n}^{\omega}\right)$$

for an equation system in $n_v$ variables and with maximum degree $d_{\max}$ reached during the Gröbner basis computation. $\omega$ denotes the linear algebra constant $2 \leq \omega < 3$. We use $\omega = 2$ for our estimates. We use the expression

$$\mathcal{C}_{\text{GB}} \coloneqq \binom{n_v + d_{\max}}{n}^{\omega}$$

directly in our complexity estimates.

| $p$ | 11 | 29 | 61 | 113 |
|---|---|---|---|---|
| $l$, $n$ | 10, 10 | 10, 10 | 10, 10 | 10, 10 |
| $s_1$, $s_2$ | 2, 2 | 2, 3 | 2, 4 | 4, 3 |
| $d_{\text{reg}}$ | 18 | 34 | 66 | 74 |
| $d_{\text{mag}}$ | 11 | 14 | 19 | 24 |
| $d_{\text{reg}} : d_{\text{mag}}$ | 1.62 | 2.43 | 3.47 | 3.08 |
| $T(s)$ | 0.09 | 1.50 | 40.00 | 1418.50 |
| $C_{\text{bit}}/2$ | 18.4 | 20.9 | 24.2 | 27.0 |

**Table 4.** Results of Gröbner basis computations on small-scale instances of a single round of `Monolith` in the CICO setting for $t = 4$ words, $u = 2$ `Bar` elements per `Bar` layer, various primes $p$, and decompositions into 2 buckets with bucket sizes $2^{s_1}$ and $2^{s_2}$. The degree of regularity $d_{\text{reg}}$ is computed under the assumption that the input system is regular, the timings of the Gröbner basis computations $T$ are given in seconds, and the estimated bit complexity $C_{\text{bit}} \coloneqq \log_2(C_{\text{GB}})$ is divided by 2 (to reflect practical runtimes). $l$ denotes the number of equations and $n$ the number of variables. $d_{\text{mag}}$ denotes the maximum degree reached during a Gröbner basis computation with the computational algebra system Magma.

**Discussion of Gröbner Basis Experiments.** The results of our Gröbner basis experiments on small-scale instances of one round of `Monolith` with $t = 4$ words and modelled as a CICO problem are depicted in Table 4. We conducted our experiments on a machine with an Intel Xeon E5-2630 v3 @ 2.40GHz (32 cores) and 378GB RAM under Debian 11 using Magma V2.26-2.

We observed a significantly faster runtime in practice than the theoretical complexity estimates indicated, even when we used $d_{\max} = d_{\text{mag}}$ in the expression $\mathcal{C}_{\text{GB}} = \binom{n_v + d_{\max}}{n}^{\omega}$. This is the reason why we chose to divide the bit complexity $\log_2(\mathcal{C}_{\text{GB}})$ by 2 and use this estimate as an indicator for the cost of computing a Gröbner basis for above algebraic model. Put differently, this is equivalent to using $\omega = 1$ in $\mathcal{C}_{\text{GB}}$. This is a highly optimistic scenario for an attacker, and we argue that even in this optimistic scenario our design is still secure against Gröbner basis attacks. We therefore conclude that using `Monolith` with 6 rounds provides ample security margin against Gröbner basis attacks.

### 5.5 Not-Applicable Attacks

We emphasize that we do not claim security of `Monolith` against zero-sum partitions [BCC11] (which can be set up via higher-order differentials [Knu94; BCD+20] and/or integral/square attacks [DKR97]). In such an attack, the goal is to find a collection of disjoint sets of inputs and corresponding outputs for the given permutation that sum to zero (i.e., satisfy the zero-sum property). Our choice is motivated by the fact that, to the best of our knowledge, it is not possible to turn such a distinguisher into an attack on the hash and/or compression function. For example, in the case of SHA-3/Keccak [Nat15; BDP+11], while 24 rounds of Keccak-$f$ can be distinguished from a random permutation using a zero-sum partition [BCC11] (that is, full Keccak-$f$), preimage/collision attacks on Keccak can only be set up for up to 6 rounds of Keccak-$f$ [GLL+20]. Indeed, the authors of Keccak-$f$ deem a 12-round version of the primitive to provide ample security margin [BDP+18]. For this reason and as already done in similar work [GKR+21; GHR+22], we ignore zero-sum partitions for practical applications.

## 6 Performance Evaluation

### 6.1 Plain Performance

In this section, we implement `Monolith` in Rust and compare its plain performance to its competitors in Table 5. Thereby, we included implementations of `Monolith` into the framework in [IAI21],[6] and also added instantiations of Poseidon [GKR+21], Poseidon2 [GKS23], and Griffin [GHR+22], with $p = 2^{64} - 2^{32} + 1$. We benchmark against these designs since Poseidon has become an unofficial standard for many zero-knowledge proof use cases, Poseidon2 being the fastest non-lookup based generic arithmetization friendly hash function in the literature so far, and Griffin being the fastest hash function in plain for the $x^{1/d}$ line of designs, which also includes *Rescue*-Prime [SAD20] and Anemoi [BBC+22].[7] We benchmark these hash function with a state size of $t = 8$ and $t = 12$ to benchmark both a sponge mode and the compression mode from Section 2.2 to have a fair comparison. Furthermore, we compare against Tip5 with its fixed state size of $t = 16$ using the implementation from [SLS+23],[8] and against Tip4′, a faster instance of Tip5 with a fixed state size $t = 12$, using the implementation from [Sal23].[9] We also compare against `Reinforced Concrete` instantiated with the scalar field of the BN254 curve [Woo+14], and against SHA3-256/SHA-256 as implemented in `RustCrypto`.[10] Finally, we compare `Monolith` to Poseidon and Poseidon2 (i.e., the fastest generic arithmetization frienldy hash functions) when instantiated with $p = 2^{31} - 1$ and state sizes of $t = 16$ and $t = 24$ (again for sponge and compression mode). All benchmarks were taken on an AMD Ryzen 9 7900X CPU (singlethreaded, 4.7 GHz).

[6] Source code is thus available at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/tree/master/plain_impls.
[7] See, e.g., https://github.com/anemoi-hash/hash_f64_benchmarks
[8] https://github.com/Neptune-Crypto/twenty-first
[9] https://github.com/Nashtare/winterfell
[10] https://github.com/RustCrypto/hashes

**Table 5.** Plain performance comparison in nano seconds ($ns$) of different hash functions. Benchmarks are given for one permutation call, i.e., hashing $\approx 500$ bits. Implemented in Rust. $^\star$ indicates an implementation without circulant MDS matrix.

| Hashing algorithm | Time for one permutation ($ns$) 2-to-1 compression | sponge |
|---|---|---|
| $p = 2^{64} - 2^{32} + 1$: | $t = 8$ | $t = 12$ |
| `Monolith`-64 | **129.9** | **210.5** |
| Poseidon | 1897.6 | 3288.7 |
| Poseidon2 | 944.6 | 1291.5 |
| Griffin | 1815.0 | 1988.4 |
| Tip5 ($t = 16$) | | 463.6 |
| Tip4′ | | 247.9 |
| $p = 2^{31} - 1$: | $t = 16$ | $t = 24$ |
| `Monolith`-31 | **210.3** | **1015.3**$^\star$ |
| Poseidon | 4478.8 | 8539.7 |
| Poseidon2 | 792.8 | 1257.4 |
| Other: | | |
| `Reinforced Concrete` (BN254) | | 1467.1 |
| SHA3-256 | | 189.8 |
| SHA-256 | 45.3 | |

**Table 6.** Plain performance comparison in nano seconds ($ns$) of different hash functions with a constant-time modular reduction and no lookup tables. Benchmarks are given for one permutation call. Implemented in Rust. $^\star$ indicates an implementation without circulant MDS matrix.

| Hashing algorithm | Time ($ns$) | |
|---|---|---|
| $p = 2^{64} - 2^{32} + 1$: | $t = 8$ | $t = 12$ |
| `Monolith`-64 | **148.5** | **230.4** |
| Poseidon | 2347.6 | 4059.1 |
| Poseidon2 | 1149.2 | 1617.9 |
| $p = 2^{31} - 1$: | $t = 16$ | $t = 24$ |
| `Monolith`-31 | **237.9** | **1120.5**$^\star$ |
| Poseidon | 4372.9 | 8538.0 |
| Poseidon2 | 840.7 | 1355.3 |

We see that `Monolith`-64 is significantly faster than any other arithmetization-oriented hash function. For example, the fastest one, i.e., Poseidon2, is slower by a factor 7.3 for $t = 8$. Tip4′, the fastest lookup table based design, is also slower by a factor of 1.9 when using `Monolith` with the compression mode, and also slower by $36\,ns$ compared to `Monolith` with the same state size $t = 12$.

Most interestingly, the performance gap between arithmetization-friendly hash functions and traditional ones is now closed, with SHA3-256 being slower than `Monolith`-64 with $t = 8$ and only faster by $21\,ns$ than `Monolith`-64 in the sponge mode with $t = 12$.

Regarding `Monolith`-31 for the 31 bit Mersenne prime field we observe that we still get a fast plain performance with $210\,ns$ for $t = 16$. This is significantly faster than Tip5 which has the same state size, but is implemented with the larger 64 bit prime field. Only for $t = 24$ we observe a slower plain performance which is due to the usage of a generic MDS matrix in the `Concrete` layer instead of an optimized circular matrix as we use for the other state sizes. However, competing designs, such as Tip5 also rely on MDS matrices and thus will suffer from the same performance loss. Despite this unoptimized linear layer one can observe that `Monolith`-31 is still faster than the fastest competitor for the same prime field and state size, i.e., Poseidon2. We will leave it for future work to find a more optimized $24 \times 24$ MDS matrix which will further speed up the plain

**Table 7.** Plain performance of each different round function in `Monolith`. Implemented in Rust. $^\star$ indicates an implementation without circulant MDS matrix.

| Operation | Time ($ns$) | | Const. Time ($ns$) | |
|---|---|---|---|---|
| $p = 2^{64} - 2^{32} + 1$: | $t = 8$ | $t = 12$ | $t = 8$ | $t = 12$ |
| `Concrete` | 19.5 | 33.6 | 19.5 | 33.6 |
| `Bricks` | 12.2 | 19.3 | 16.0 | 21.8 |
| `Bars` | 10.4 | 12.9 | 10.4 | 12.9 |
| $p = 2^{31} - 1$: | $t = 16$ | $t = 24$ | $t = 16$ | $t = 24$ |
| `Concrete` | 31.8 | 138.1$^\star$ | 31.9 | 138.1$^\star$ |
| `Bricks` | 17.0 | 21.7 | 17.0 | 21.7 |
| `Bars` | 8.4 | 12.0 | 8.4 | 12.0 |

performance of `Monolith`-31. In the meantime we point out that using a $t = 16$ sponge, where 8 field elements are reserved for the inner and outer part respectively allows to absorb 512 bit of data with two permutation calls. In that sense one can use this sponge instead of a $t = 24$ sponge at the cost of evaluating two $t = 16$ permutations, leading to a hashing performance of $\approx 420\,ns$, which is significantly faster than any competitor for this specific prime field.

Another advantage of `Monolith` over Tip5, Tip4′, and `Reinforced Concrete` is that its plain performance does not rely on lookup tables and its structure allows for constant-time implementations without significant performance loss. The binary $\chi$-like layer can be efficiently implemented using a vectorized implementation that does not require an explicit (de-)composition, while unrolling the lookup-tables containing repeated power maps in `Reinforced Concrete`, Tip5, and Tip4′ adds considerable workload to the computation. Thus, the overhead of going to a constant-time implementation only consists of supporting constant-time prime field arithmetic for `Monolith`, which can help in efficiently preventing side-channel attacks such as the ones proposed in [TBP20]. To this end we rewrite the fast modular reduction to be constant-time and replace the lookup-tables in the `Bars` layer with its arithmetic description to instantiate a constant-time version of `Monolith`. We give these benchmarks in Table 6.

We observe, that using a constant-time modular reduction leads to a slight slowdown of all benchmarked designs. However, the resulting runtimes are still significantly faster than the non-constant-time runtimes of traditional arithmetization friendly hash functions, such as Poseidon and Griffin, and the variable-time version of Tip4′ for $t = 8$ and $t = 12$. Moreover, a constant time `Monolith`-64 in compression mode is still faster than SHA3-256 for $t = 8$ (even if we acknowledge the different security margin of the two constructions).

Finally, for the sake of completeness, we give the runtime of each part of the `Monolith` permutation for both a constant- and variable-time version in Table 7.

### 6.2 Performance in Proof Systems

A modern zero-knowledge proof system defines, among other things, *arithmetization rules* for the computations it attempts to prove. Most new proof systems support the *Plonkish* arithmetization. Without loss of generality, its rules can be described as follows.

- The entire computation is represented as a sequence of polynomial computations over input and intermediate data and table relations over data tuples.
- All input, output, and intermediate variables are placed into a *witness matrix* $W$ with $n$ columns and $m$ rows.
- The data in each row is restricted by polynomial equations determining the values and precise computations being used. One of these generic equations of degree 2, used in the original Plonk paper [GWC19], is

$$a_i x_1 x_2 + b_i x_3 + c_i x_4 + d_i = 0,$$

  where $a_i, b_i, c_i, d_i$ are public constants for the $i$-th row. The Plonkish arithmetization allows for different tradeoffs between the number of columns or variables being used and the resulting degrees.

– Additionally, various tuples within a row may be constrained to a table entry. This can be defined as $(x_1, x_2, x_3) \in \mathfrak{T}$, where $\mathfrak{T}$ is a predefined table.

It can be seen that there can be many valid ways to arithmetize a particular computation.

The influence of the arithmetization parameters on the prover cost is not immediately clear, and for a precise comparison it is necessary to benchmark on the target proof system. Nevertheless, it can be seen that the dominant prover work is to construct $m$ polynomials of degree $n$ for the witness columns and to prove that they satisfy each of the polynomial equations. The total work can then be estimated as an element in $\mathcal{O}(d \cdot n \cdot m)$, where $d$ is the maximum degree of a row polynomial relation. The cost of using table lookups for FRI-based schemes is currently equivalent to the use of a single polynomial of degree $t = \max\{n, \text{size}(\mathfrak{T})\}$, i.e., it is not recommended to use a table with more rows than in the witness matrix.

In this section we give possible arithmetizations for translating `Monolith` into a set of Plonkish and R1CS constraints. Our Plonkish arithmetization is designed to accommodate lookup constraints capable of efficiently looking up 8-bit values. If the proof system is able to use larger tables (e.g., 16-bit ones), then multiple lookup constraints can be combined into just one larger constraints, reducing the total number of constraints.

**Plonkish.** We suggest the following arithmetization for `Monolith`.

– Each composition `Concrete` $\circ$ `Bricks` is described with $t$ polynomial equations of degree 2.
– Each `Bar` in the `Bars` layer is described as follows.
  • We describe the application of $m$ individual S-boxes with $m$ lookup constraints $(x_1, y_1)$, $(x_2, y_2), \ldots, (x_m, y_m)$. These lookup constraints also include the range checks for each input.
  • For $p_{\texttt{Goldilocks}}$, we restrict all `Bar` inputs to the field, enforcing that either the least significant 32 bits are 0 or the most significant bits are not all 1, i.e.,

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

  • For $p_{\texttt{Mersenne}} = \texttt{0x7fffffff}$ we only need to check that the combined values are not equal to $p$, which is equivalent to them not being $2^8 - 1$ (three) or $2^7 - 1$ (one), i.e.,

$$(x_4 + x_3 + x_2 + x_1 - 2^7 - 3 \cdot 2^8 + 4) \cdot z' = 1.$$

    The range checks of the lookup constraints will take care for verifying correctness otherwise.
  • We require the correct composition and decomposition, i.e.,

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i, \quad y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i.$$

  • Apart from $2m$ lookup variables per `Bar` we define
    ∗ $t - u$ variables for the `Bars` layer that correspond to the identity function application, and
    ∗ $t$ input and output variables to `Bricks` so that the nonlinear constraints have few terms.

    All the other trace elements are linear functions of those. Altogether, we have $6(2 \cdot 8 \cdot 4 + t + 2t - 4) = 18t + 360$ variables for the $p_{\texttt{Goldilocks}}$ case and $6(2 \cdot 4 \cdot 8 + t + 2t - 8) + t = 18t + 336$ variables for the $p_{\texttt{Mersenne}}$ case.
  • In the $p_{\texttt{Goldilocks}}$ case we have $m = 8$ and 4 parallel `Bar` functions, hence 32 lookups, $t$ degree-2 constraints, and $t$ linear equations per round. In the $p_{\texttt{Mersenne}}$ case this results in $8m$ lookups, $t$ degree-2 constraints, and $t$ linear equations.

*Tip5 Simple Arithmetization.* The Tip5 function applies four 64-bit S-boxes with lookups per round, so 32 8-bit lookups per round. It also uses 12 degree-7 power functions per round. We allocate variables for the inputs and outputs of the power functions in addition to 64 lookup variables per round.

Similarly, the Tip4' function also applies 32 8-bit lookups per round to the smaller state. However, it uses 8 degree-7 power functions per round, proportionally reducing the number of variables.

The Poseidon2 function (as well as Poseidon which has the same number of rounds and the same arithmetization) with $t = 12$ defined for $p_{\texttt{Goldilocks}}$ has 8 full and 22 partial rounds, thus 118 degree-7 functions in total. We allocate variables for all inputs and outputs of the S-boxes, and link the others via linear equations.

In Table 8 we compare the (non-optimized) arithmetization of `Monolith` with the ones of other 64-bit designs. To achieve a fair comparison, we do not apply any constraint or witness optimization but try to follow the same approach. We see that both the number of lookups and constraints in `Monolith` is slightly larger than in Tip5 and Tip4', but the constraint degree is smaller by the factor of 3.5, which should result in an overall decrease of the prover time by a factor of at least 2 (estimated as area-degree product). This is reasonable since Tip5 and Tip4' are able to process more field elements with a permutation call. Interestingly, Poseidon and Poseidon2 appear somewhere in between. Their bigger number of constraints is compensated by a smaller state. Again, we stress that these numbers are derived from non-optimized arithmetizations and are subject to change.

| 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 |

**Table 8.** Plonkish arithmetization comparison for various 64-bit schemes. The numbers are for a single permutation.

**Multiround Constraints for `Monolith`.** We consider $p = p_{\texttt{Goldilocks}}$ and $t = 12$. When implementing both `Monolith` and Tip5 in a single gate, we can immediately observe various similarities. For example, considering 8-bit lookups, the number of lookups is almost the same, with Tip5 using slightly fewer ones due to its lower number of rounds (note that both permutations use four lookup words per round). Moreover, the number of necessary columns is similar in a round-based approach.

The major advantage of `Monolith` becomes apparent after considering the degree of the constraints. Indeed, while Tip5 uses a maximum degree of 7 (which is the smallest integer $d$ such that $\gcd(p_{\texttt{Goldilocks}} - 1, d) = 1$), `Monolith` uses a maximum degree of only 2. Not only does this lead to more efficient constraints, but it allows for different tradeoffs. For example, consider $p = p_{\texttt{Goldilocks}}, t = 12$ and a state after the `Concrete` layer defined by 12 variables $w_1^{(1)}, \dots w_{12}^{(1)}$. After the subsequent application of `Bars`, we add 4 new variables $w_1^{(2)}, \dots, w_4^{(2)}$ for the state elements modified by the lookup table. We now apply `Bricks` and then `Concrete` to the state. Note that describing the state in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}$ after these transformations results in degree-2 constraints (ignoring the table lookups), since only one `Bricks` layer has been applied. Hence, we may now choose to only add 4 new variables $w_1^{(3)}, \dots, w_4^{(3)}$ after the application of the last `Concrete` layer at the positions of the table lookups. After the next `Bars` layer, the state is defined by 8 polynomial equations in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}$ of degree 2 and by the 4 new variables $w_1^{(4)}, \dots, w_4^{(4)}$ resulting from the table lookups. After applying the next `Bricks` and `Concrete` layers, we arrive at a state defined by 12 polynomial constraints in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}, w_1^{(4)}, \dots, w_4^{(4)}$ of degree 4. A graphical overview of this approach is shown in Fig. 4.

As a result, with degree-4 constraints we can save $t - u$ trace elements in each pair of rounds, where $u$ is the number of `Bar` applications in the `Bars` layers. This allows us to achieve a slimmer row with even fewer columns. We point out that this advantage of `Monolith`'s low degree also applies

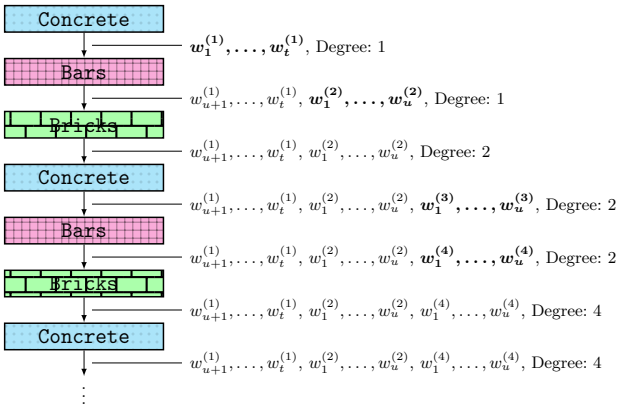


**Fig. 4.** Variables (or trace elements) when using `Monolith` with degree-4 constraints. Newly added variables are emphasized in **bold** and the degree indicates the maximum degree of the polynomial equations describing the corresponding state in the given variables.

in a similar fashion when comparing to other hash functions which use $x^d$, such as Poseidon, Poseidon2, *Rescue*, Griffin, Anemoi, and many more.

**R1CS.** It is possible, though more expensive, to implement `Monolith` in legacy proof systems that only support R1CS equations without any table lookups. In contrast to `Reinforced Concrete`, our design admits a reasonably small R1CS representation described in the following.

- We use $t - 1$ constraints to generate equations for `Bricks`.
- For `Bars`, we decompose each element that goes into a `Bar` into bits thus using one constraint per `Bar` for the actual decomposition plus $\log_2(p) \cdot \#\texttt{Bar}$ constraints for ensuring that the bits are either 0 or 1. Then each output bit of `Bar` requires 3 multiplications (2 for `AND` and 1 for `XOR`) for the 8-bit S-box and 2 multiplications for the 7-bit one as used in `Monolith`-31. By combining the composition constraints with the following bricks layer we get 1028 constraints for `Monolith`-64 and 944 constraints for `Monolith`-31 per `Bars`.
- The `Concrete` layer can be included in the constraints of `Bricks` and `Bars`.
- In total, $R$ rounds require $R \cdot (1027 + t)$ R1CS constraints for `Monolith`-64 and $R \cdot (943 + t)$ for `Monolith`-31.

## Acknowledgments

Lorenzo Grassi is partially supported by the German Research foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa – 39078197. Roman Walch is supported by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG.

## References

[AAE+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). Available at https://eprint.iacr.org/2019/426.pdf, pp. 1–45 (cit. on p. 3).

[AD18] Tomer Ashur and Siemen Dhooghe. "MARVELlous: a STARK-Friendly Family of Cryptographic Primitives". In: *IACR Cryptol. ePrint Arch.* (2018) (cit. on p. 3).

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, et al. "Feistel Structures for MPC, and More". In: *ESORICS 2019*. Vol. 11736. LNCS. Available at https://eprint.iacr.org/2019/397.pdf. 2019, pp. 151–171 (cit. on p. 3).

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *ASIACRYPT 2016*. Vol. 10031. LNCS. Available at https://eprint.iacr.org/2016/492.pdf. 2016, pp. 191–219 (cit. on p. 3).

[AKM+22] Jean-Philippe Aumasson, Dmitry Khovratovich, Bart Mennink, and Porçu Quine. *SAFE (Sponge API for Field Elements) - A Toolbox for ZK Hash Applications*. https://eprint.iacr.org/2023/522. 2022 (cit. on p. 7).

[BBC+22] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, et al. *New Design Techniques for Efficient Arithmetization-Oriented Hash Functions:Anemoi Permutations and Jive Compression Mode*. Cryptology ePrint Archive, Paper 2022/840. https://eprint.iacr.org/2022/840 – accepted at CRYPTO 2023. 2022 (cit. on pp. 3, 7, 24).

[BBH+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *CRYPTO (3)*. Vol. 11694. LNCS. Available at https://www.iacr.org/archive/crypto2019/11694020l/11694020l.pdf. Springer, 2019, pp. 701–732 (cit. on p. 3).

[BBL+22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. "Algebraic Attacks against Some Arithmetization-Oriented Primitives". In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022). Available at https://tosc.iacr.org/index.php/ToSC/article/view/9850/9350, pp. 73–101 (cit. on p. 11).

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. "Higher-Order Differential Properties of Keccak and *Luffa*". In: *FSE 2011*. Vol. 6733. LNCS. Available at https://eprint.iacr.org/2010/589.pdf. 2011, pp. 252–269 (cit. on p. 24).

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, et al. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems". In: *CRYPTO 2020*. Vol. 12172. LNCS. Available at https://eprint.iacr.org/2020/188.pdf. 2020, pp. 299–328 (cit. on p. 24).

[BDP+07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *Sponge functions*. In: Ecrypt Hash Workshop 2007, http://www.csrc.nist.gov/pki/HashWorkshop/PublicComments/2007_May.html. 2007 (cit. on p. 7).

[BDP+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction". In: *EUROCRYPT 2008*. Vol. 4965. LNCS. Available at https://keccak.team/files/SpongeIndifferentiability.pdf. 2008, pp. 181–197 (cit. on p. 7).

[BDP+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Note on zero-sum distinguishers of Keccak-f*. Available at https://keccak.team/files/NoteZeroSum.pdf. 2011 (cit. on p. 24).

[BDP+18] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. "KangarooTwelve: Fast Hashing Based on Keccak-p". In: *ACNS*. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 400–418 (cit. on p. 24).

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. Avaiable at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf. 2005 (cit. on p. 4).

[BFM+16] Thierry P. Berger, Julien Francq, Marine Minier, and Gaël Thomas. "Extended Generalized Feistel Networks Using Matrix Representation to Propose a New Lightweight Block Cipher: Lilliput". In: *IEEE Trans. Computers* 65.7 (2016). Preliminary version available at https://hal.science/hal-00913881v1/file/Berger_Minier_Thomas_Extended_Generalized_Feistel_Networks_using_Matrix_Representation.pdf, pp. 2074–2089 (cit. on p. 11).

[BFS+05] Magali Bardet, Jean-Charles Faugere, Bruno Salvy, and Bo-Yin Yang. "Asymptotic behaviour of the index of regularity of quadratic semi-regular polynomial systems".

In: *The Effective Methods in Algebraic Geometry Conference (MEGA)*. Available at https://www-polsys.lip6.fr/~jcf/Papers/BFS05.pdf. 2005, pp. 1–14 (cit. on p. 22).

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *CRYPTO 1990*. Vol. 537. LNCS. Available at http://www.cs.bilkent.edu.tr/~selcuk/teaching/cs519/Biham-DC.pdf. 1990, pp. 2–21 (cit. on p. 15).

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In: *EUROCRYPT (1)*. Vol. 12105. LNCS. Available at https://eprint.iacr.org/2019/1076.pdf. Springer, 2020, pp. 769–793 (cit. on pp. 3, 4).

[Dae95] Joan Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. Doctoral Dissertation. Available at https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf. 1995 (cit. on pp. 5, 9, 13).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *FSE 1997*. Vol. 1267. LNCS. Available at https://link.springer.com/content/pdf/10.1007/BFb0052343.pdf. 1997, pp. 149–165 (cit. on p. 24).

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Available at https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf. Springer, 2002 (cit. on p. 6).

[Fau02] Jean Charles Faugère. "A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5)". In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. Association for Computing Machinery, 2002, pp. 75–83 (cit. on p. 22).

[FGL+93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. "Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering". In: *J. Symb. Comput.* 16.4 (1993), pp. 329–344 (cit. on p. 22).

[GHR+22] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. *Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications*. Cryptology ePrint Archive, Paper 2022/403. https://eprint.iacr.org/2022/403 – accepted at CRYPTO 2023. 2022 (cit. on pp. 3, 7, 21, 24).

[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: *ACM CCS*. Available at https://eprint.iacr.org/2021/1038.pdf. 2022, pp. 1323–1335 (cit. on pp. 3, 10).

[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 (cit. on pp. 3, 24).

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. *Poseidon2: A Faster Version of the Poseidon Hash Function*. Cryptology ePrint Archive, Paper 2023/323. https://eprint.iacr.org/2023/323 – accepted at AFRICACRYPT 2023. 2023 (cit. on pp. 3, 7, 24).

[GLL+20] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. "Practical Collision Attacks against Round-Reduced SHA-3". In: *J. Cryptol.* 33.1 (2020), pp. 228–270 (cit. on p. 24).

[GOP+22] Lorenzo Grassi, Silvia Onofri, Marco Pedicini, and Luca Sozzi. "Invertible Quadratic Non-Linear Layers for MPC-/FHE-/ZK-Friendly Schemes over $\mathbb{F}_p^n$ – Application to Poseidon". In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022), pp. 20–72 (cit. on p. 3).

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *EUROCRYPT (2)*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 305–326 (cit. on p. 4).

[GW20] Ariel Gabizon and Zachary J. Williamson. "plookup: A simplified polynomial protocol for lookup tables". In: *IACR Cryptol. ePrint Arch.* (2020) (cit. on p. 3).

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019 (cit. on pp. 3, 4, 26).

[Hab23] Ulrich Haböck. *Brakedown's expander code*. Cryptology ePrint Archive, Paper 2023/769. https://eprint.iacr.org/2023/769. 2023 (cit. on p. 4).

[HLN23] Ulrich Haböck, Daniel Lubarov, and Jacqueline Nabaglo. *Reed-Solomon Codes over the Circle Group*. Cryptology ePrint Archive, Paper 2023/824. https://eprint.iacr.org/2023/824. 2023 (cit. on p. 4).

[IAI21] IAIK. *Hash functions for Zero-Knowledge applications Zoo*. https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo. IAIK, Graz University of Technology. Aug. 2021 (cit. on p. 24).

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. "MinRoot: Candidate Sequential Function for Ethereum VDF". In: *IACR Cryptol. ePrint Arch.* (2022) (cit. on p. 3).

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE 1994*. Vol. 1008. LNCS. 1994, pp. 196–211 (cit. on pp. 17, 24).

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography: Two Sides of One Tapestry*. Springer US, 1994, pp. 227–233 (cit. on p. 18).

[MRS+09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. "The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl". In: *FSE*. Vol. 5665. LNCS. Springer, 2009, pp. 260–276 (cit. on p. 17).

[Nat15] National Institute of Standards and Technology. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". In: *Federal Information Processing Standards Publication (FIPS)* (202 2015) (cit. on pp. 3, 12, 24).

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *CT-RSA*. Vol. 3860. Lecture Notes in Computer Science. Springer, 2006, pp. 1–20 (cit. on p. 4).

[Pag02] Daniel Page. *Theoretical use of cache memory as a cryptanalytic side- channel*. 2002 (cit. on p. 4).

[Pol22a] Polygon. *Introducing Plonky2*. 2022 (cit. on p. 4).

[Pol22b] Polygon. *Plonky2: Fast Recursive Arguments with PLONK and FRI*. 2022 (cit. on p. 4).

[Pol23] Polygon. *Plonky3*. 2023. URL: https://github.com/Plonky3/Plonky3 (visited on 06/12/2023) (cit. on p. 4).

[RIS23] RISC Zero. *RISC Zero : General-Purpose Verifiable Computing*. 2023 (cit. on p. 4).

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. *Rescue-Prime: a Standard Specification (SoK)*. Cryptology ePrint Archive, Report 2020/1143. 2020 (cit. on pp. 3, 24).

[Sal23] Robin Salen. "Two additional instantiations from the Tip5 hash function construction". In: *https://toposware.com/paper_tip5.pdf* (2023) (cit. on p. 24).

[SC21] Abdurrashid Ibrahim Sanka and Ray C. C. Cheung. "A systematic review of blockchain scalability: Issues, solutions, analysis and future research". In: *J. Netw. Comput. Appl.* 195 (2021), p. 103232 (cit. on p. 3).

[SLS+23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, Bobbin Threadbare, and Al-Kindi. *The Tip5 Hash Function for Recursive STARKs*. Cryptology ePrint Archive, Paper 2023/107. https://eprint.iacr.org/2023/107. 2023 (cit. on pp. 3, 8, 24).

[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 (cit. on pp. 4, 26).

[Woo+14] Gavin Wood et al. *Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper.(2014)*. 2014 (cit. on p. 24).

[YMT97] A. M. Youssef, S. Mister, and S. E. Tavares. "On the Design of Linear Transformations for Substitution Permutation Encryption Networks". In: *School of Computer Science, Carleton University*. 1997, pp. 40–48 (cit. on p. 8).

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive, Paper 2022/1565. https://eprint.iacr.org/2022/1565. 2022 (cit. on p. 3).

[ZMI89] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses". In: *CRYPTO 1989*. Vol. 435. LNCS. 1989, pp. 461–480 (cit. on pp. 5, 7).

# A Fast Reduction for Primes of the Form $\phi^2 - \phi + 1$ and $2^\rho - 1$

## A.1 Fast Reduction for Primes of the Form $\phi^2 - \phi + 1$

Here we describe the fast reduction modulo a prime number of the form $\phi^2 - \phi + 1$. Note that this includes $p = 2^{64} - 2^{32} + 1$, where $\phi = 2^{32}$. We focus on the case of a multiplication, where two $n$-bit inputs result in an output of at most $2n$ bits.

Given $\mathbb{F}_p$ for $p = \phi^2 - \phi + 1$, it follows that

$$\phi^2 = \phi - 1 \implies \phi^3 = \phi^2 - \phi = -1.$$

Now, let us write a value $x$ to be reduced as

$$x = x_0 + \phi^2 x_1 + \phi^3 x_2,$$

where $x_0 \in \mathbb{Z}_{2^n}$ and $x_1, x_2 \in \mathbb{Z}_{2^{n/2}}$. Then

$$x = x_0 + (\phi - 1)x_1 - x_2 \pmod{p},$$

where note that $\log_2(x_0 + (\phi - 1)x_1 - x_2) \approx \log_2(p)$. This reduction can be computed using only a small number of additions and subtractions.

## A.2 Fast Reduction for Primes of the Form $2^\rho - 1$

Here we describe the fast reduction modulo a prime number of the form $2^\rho - 1$ which includes $p = 2^{31} - 1$. We focus on the case of a multiplication, where two $\rho$-bit inputs result in an output of at most $2\rho$ bits.

Given $\mathbb{F}_p$ for $p = 2^\rho - 1$, it follows that $2^\rho = 1 + p$. Now, let us write a value $x$ to be reduced as

$$x = x_0 + 2^\rho x_1,$$

where $x_0 \in \mathbb{Z}_{2^\rho}$ and $x_1 \in \mathbb{F}_p$. Then

$$x = x_0 + x_1 + \underbrace{(2^\rho - 1) \cdot x_1}_{=0 \pmod{p}} = x_0 + x_1 \pmod{p}.$$

This reduction can be computed using only a small number of additions and binary shifts.