Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations

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. Hash functions are a crucial component in incrementally verifiable computation (IVC) protocols and applications. Among those, recursive SNARKs and folding schemes require hash functions to be both fast in native CPU computations and compact in algebraic descriptions (constraints). However, neither SHA-2/3 nor newer algebraic constructions, such as Poseidon, achieve both requirements.

In this work we overcome this problem in two steps. First, for certain prime field domains we propose a new design strategy called `Kintsugi`, which explains how to construct nonlinear layers of high algebraic degree which allow fast native implementations and at the same time also an efficient circuit description for zero-knowledge applications. Then we suggest another layer, based on the Feistel Type-3 scheme, and prove wide trail bounds for its combination with an MDS matrix.

Finally, we propose a new permutation design named `Monolith` to be used as a sponge or compression function. It is the first arithmetization-oriented function with a native performance comparable to SHA3-256. At the same time, it outperforms Poseidon in a circuit using the Merkle tree prover in the Plonky2 framework. Contrary to previously proposed designs, `Monolith` also allows for efficient constant-time native implementations which mitigates the risk of side-channel attacks.

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 [GWC19; ZGK+22; KST22; BC23] and low-level

primitives [AGR+16; AAE+20; GKR+21] have been designed and published recently, in an attempt to increase the performance in this setting. The emergence of folding techniques and recursive SNARKs (incrementally verifiable computation, or IVC [Val08]) make it possible to efficiently prove the integrity of complex computations. Proofs with 2^{27} steps have been recorded[1] whereas SNARK-based verifiable delay functions (VDFs) might require proving up to 2^{40} operations [KMT22]. A single IVC operation is typically a compact arithmetic computation (polynomial) in a certain prime field or an assertion to some low-degree polynomial predicate. 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 used not only for data integrity checks, but also to instantiate commitment schemes, authenticated encryption [PSS19; CFG+22], non-interactive proofs based on the Fiat–Shamir transform, and many other techniques.

Hash Functions in IVC Applications. For "classical" applications of hash functions, general-purpose standard choices like SHA-2 or SHA-3 are usually not the bottleneck.[2] However, the situation is different in the IVC applications mentioned above. For hashing and membership proofs in ZK, with folding schemes [KST22; KS23; BC23] and private mixers like Tornado being an example [PSS19], the size of hash function as an *arithmetic circuit over a prime field* is more important as a cost metric than the "native" software performance (e.g., on a x86 architecture). New hash functions have tried to bridge this gap [AGR+16; AAE+20; GHR+23; GKR+21; SAD20; BBC+23].

Another example is using hash functions as a commitment tool in IVC frameworks where the underlying commitment scheme may not be homomorphic – with STARKs being a notable example [BBH+19]. With a prover and a verifier engaging in commit-open protocols (again, over certain prime fields), this use case requires to efficiently construct an entire Merkle tree in a prime field domain over large amounts of data. So far, though, the computations were performed natively on x86 hardware and not (yet) inside a circuit. Here, classical hash functions have been used up until recently.

Both cases appear in recursive schemes, in particular in recursive STARKs [COS20], which are an attractive IVC concept due to relatively little overhead and the possibility of parallelism for large or long computations. These schemes are used in an increasing number of applications, including zero-knowledge virtual machines [22a; 22b; Zha22] and decentralized signature aggregation [But22] protocols as notable examples. In recursive STARKs the computation and its proof are broken into chunks $C_1, C_2, \ldots, C_k$ such that the proof π_i certifies that chunks from C_1 to C_i are computed correctly using the previous proof π_{i-1} and a proof of computing C_i. On each recursion step a prover computes a Merkle tree over the witness data and then proves some tree openings in a circuit. Thus, the *same* hash function is used in the circuit and in the native computation. In

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

[2] Newer choices that are faster by a small factor do exist [ANW+13; BDP+18].

this scenario, up to 90% of a prover's computation may be spent on the hash function call and proofs [COS20; RIS23a], and a construction of a function that excels in both areas is a crucial open problem.

Relevant IVC Techniques: Lookups and Small Domains. Two major developments in IVC have helped us in this work. The first one is the lookup technique. Starting with Plookup, the IVC operations include not only arithmetic expressions and predicates but also lookup statements of form $a \in T$, where T is a table available to the verifier [GW20; PH23; STW23]. Depending on the polynomial commitment used within IVC, the table may be preprocessed [ZBK+22; ZGK+22; EFG22], so that in the former case only the number of lookups contribute to the prover cost, whereas in the latter case the table size itself is the minimal cost. STARKs use non-homomorphic FRI commitments and thus belong to the second group. The lookup technique not only reduced the cost of traditional hash functions in circuits[3] but also allowed for cheap transformations of high algebraic degree [GKL+22; SLS+23].

The second improvement is purely technical but nevertheless vital for the performance. It consists of using small prime fields of ≤ 64 bits with primes of special forms like $2^k - 1$ [Pol22; Pol23; RIS23b], which gained special attention for high performance of arithmetic operations in the field. STARKs [BBH+19] can use them since they do not require a group where the discrete logarithm problem is assumed to be hard. The performance growth is significant: Switching to an efficient 64-bit field improves the performance by a factor of up to 10 for the Poseidon hash function [GKS23]. An important feature of these domains is that the modular reduction can be implemented with mere additions and bit shifts, which are vectorizable on modern CPU architectures and are much faster than their counterparts in large prime fields. There are also various works in the recent literature discussing smaller primes for IVC applications [HLN23; Hab23].

1.2 Our Contributions

We approach the problem of creating a fast and circuit-friendly hash function in several steps. First we summarize the technical ideas of the new design, and then we introduce the construction of the new hash function `Monolith`.

Efficient Nonlinearity and Compact Circuits over Prime Fields. Our first main contribution is a generic design of components over certain prime fields $\mathbb{F}_p$, which, on one hand, can be implemented with just a few (and possibly vector) constant-time instructions on the x86 architecture, and on another hand can be written as a small circuit over $\mathbb{F}_p$. This strategy, which we call `Kintsugi`, is an evolution of the ideas behind the `Reinforced Concrete` [GKL+22] and Tip5 [SLS+23] components. First, an element from a bigger field is efficiently split into smaller bitarrays, which is possible due to the form of the prime. Then

[3] https://zcash.github.io/halo2/design/gadgets/sha256/table16.html

we apply *constant-time S-boxes*, which are instantiated by Daemen's χ function and similar ones [Dae95] that can be implemented in a batch using fast vector instructions, or as lookup tables in circuits. Finally, the outputs are assembled back to a field element with no overflow or collision, which is asserted in circuits with minimal overhead.

Low-Degree Components with Provable Differential Bounds. Our second contribution is a concept of using a Feistel Type-3 [ZMI89] function together with an MDS layer. It is offered as a replacement to the power function x^d in Poseidon [GKR+21] and similar constructions. The advantage is that we can use faster squarings x^2 instead of more expensive (as d must be coprime with $p-1$) power functions over $\mathbb{F}_p$, and simultaneously obtain predicates of low degree in circuits. Whereas the Feistel layer alone is known to have weak diffusion, we show that together with an MDS it comes close to a regular SPN.

To the best of our knowledge, we are the first to prove the results on the differential properties of the component using a strategy analogous to the wide trail design [DR02]. In particular, we prove lower bounds on the number of active nonlinear functions in trails. Similar to extended generalized Feistel networks introduced in [BMT13], we believe that this result and its possible extension to Feistel structures of other types may be useful in the design of any symmetric primitive, not only for arithmetization-friendly schemes, but also in the case of more classical use cases (as already happened for the Lilliput cipher [BFM+16]).

`Monolith`: Fast, Constant-Time, ZK-Oriented Hashing. The combination of such techniques leads us to the design of `Monolith`[4], a family of permutations which are both efficient in native and inside of circuits and can be turned into hash functions and other permutation-based schemes.

Construction of `Monolith`. Our scheme consists of a few rounds, each using the following three components.

- The first one is `Bricks` (Section 4.3), which is instantiated with a Feistel Type-3 construction with square mappings.
- The second component is `Concrete` (Section 4.4), which is the multiplication with a circulant MDS matrix. Together with `Bricks` it provides the diffusion necessary to protect against statistical attacks.
- Finally, the third component is `Bars` (Section 4.5), which is based on the `Kintsugi` outlined above. We prove that each such `Bar` operation has a high degree and provides high security against algebraic attacks. The `Bar` function is applied only to a few field elements in each round.

The combination of these three components provides security against statistical and algebraic attacks while allowing for an efficient implementation. Our initial

[4] A *monolithic* building is a seamless structure where components are intimately fused in order to provide the most secure and robust construction.

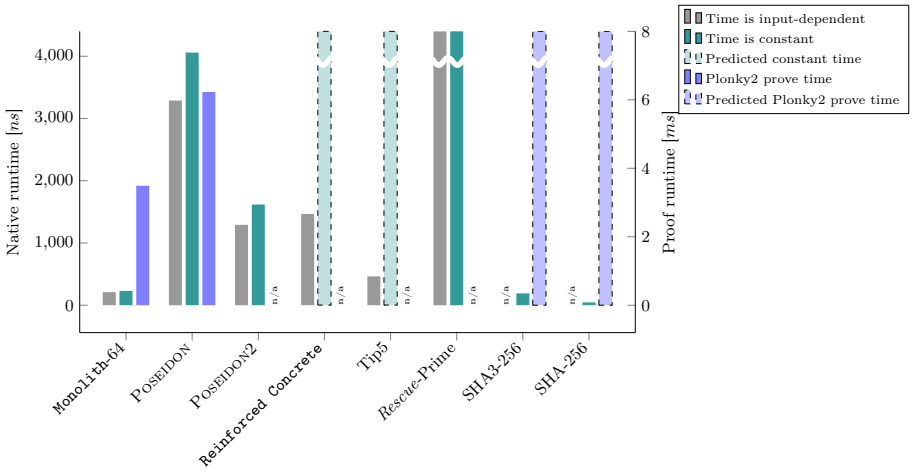

Fig. 1. Runtime comparison of different hash functions. The benchmarks are from Table 3 and the numbers for `Monolith`-64, Poseidon, and Poseidon2 are taken for the 64-bit prime field and a state size of $t = 12$ (sponge mode). Proof times are benchmarks for a proof of preimage knowledge (Table 5). Numbers for SHA3-256 and SHA-256 are extrapolated from a `circom` implementation using R1CS [Bal23].

analysis has found a 3-round attack on a weakened version, and also suggests that all potential attacks should stop at at most 4 rounds. Since improvements are expected (we encourage third-party cryptanalysis), we set the number of rounds uniformly to 6. Details can be found in Section 5.

Performance Evaluation. We give an extensive comparison between our new proposal and its competitors in Section 6. Our benchmarks confirm that the native performance of `Monolith` is comparable to SHA-3, which makes it the first circuit-friendly compression function achieving this goal. At the same time, `Monolith` is efficient within IVC 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, and it can also be reasonably used in proof systems without lookup arguments. A quick overview of the performance numbers is given in Fig. 1. We focus on the Plonky2 proving system since it is currently one of the most popular ones for FRI-based proofs.

Further, compared to Tip5, `Monolith` is around twice as fast and gives the user more freedom regarding the choice of the prime number. It can even be used with prime fields as low as 31 bits, which is a setting recently considered in applications and various proving frameworks [RIS23b] due to advantageous implementation characteristics. Moreover, compared to the widely used Poseidon permutation, `Monolith` shows a native 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, leading to a faster

prover and verifier performance compared to Poseidon when implemented in the FRI-based Plonky2 proof system [Pol22] (see Table 5).

2 Fast and Circuit-Friendly Functions over $\mathbb{F}_p$

We suggest a generic strategy to create nonlinear components over $\mathbb{F}_p$ that are efficient in native, constant-time, and circuit implementations.

2.1 The `Kintsugi` Design Strategy

When working over $\mathbb{F}_p$, informally, we cannot just split a field element into smaller chunks, process them independently, and then reassemble. This is due to the fact that the field size is a prime and thus cannot be represented as a product of smaller domains.

To solve this problem, we present a generic strategy for specifically chosen prime numbers. Elements of it can be found in earlier works on `Reinforced Concrete` [GKL+22] and Tip5 [SLS+23]. The main principles are the following.

1. Assume we work in a prime field where the prime is a sum of just a few (possibly negative) powers of two, such as $p_{\texttt{gen1}} = 2^n - 1$ or $p_{\texttt{gen2}} = 2^n - 2^\eta + 1$.
2. Split the integer form of a field element into chunks according to carefully chosen boundaries aligned with these powers of two (more details to follow) and such that the resulting (smaller) chunks fit a lookup table in a ZK circuit.
3. Identify the combination of chunk values that never appear due to the fact that p is not a power of two.
4. Design intra-chunk transformations $\mathcal{S}_i$ such that
   - impossible chunk combinations never appear (this is, e.g., done by making some chunk values fixed points), and
   - they can be implemented in constant time, for example with an AndRX (AND-rotation-XOR) transformation [AJN14].
5. Combine the chunks back into a large element, after a possible shuffle (note that only the shuffles that guarantee that the output element is in the field are possible).

We call this strategy `Kintsugi`.[5] An illustration is shown in Fig. 2. More formally, it can be defined as

$$x \mapsto \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x). \tag{1}$$

with the following components.

[5] Kintsugi is the Japanese art of repairing broken pottery by mending the areas of breakage with lacquer dusted or mixed with e.g. powdered gold. Here, we break the state and we recombine it after applying a particular function to each small chunk.

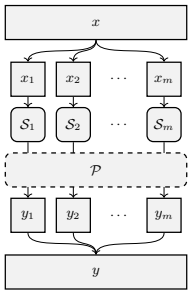

Fig. 2. The `Kintsugi` strategy, where $\mathcal{S}_i(2^{s_i-1}) = 2^{s_i-1}$, $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$ if p is of the form $p_{\texttt{gen2}}$, and $\mathcal{P}$ denotes a potential shuffling operation applied to the vector $(\mathcal{S}_1(x_1), \mathcal{S}_2(x_2), \ldots, \mathcal{S}_m(x_m))$.

Decomposition $\mathcal{D}$. The decomposition $\mathcal{D}$ decomposes the original field element $x \in \mathbb{F}_p$ into $m > 1$ smaller elements $x'_1, x'_2, \ldots, x'_m$, such that

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

over integers, where $x'_i \in \mathbb{Z}_{2^{s_i}} \equiv \mathbb{F}_{2^{s_i}}$ and $\sum s_i = n = \lceil \log_2(p) \rceil$, i.e., we apply binary decomposition. If $p = p_{\texttt{gen2}}$, we additionally require that $s_1 + s_2 + \cdots + s_l = \eta$ for some l. Equivalently, $x_i := (x \gg (s_1 + s_2 + \cdots + s_{i-1})) \odot (2^{s_i} - 1)$.

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

$$\mathcal{S}(x'_1, x'_2, \ldots, x'_m) = \mathcal{S}_1(x'_1) \mid\mid \mathcal{S}_2(x'_2) \mid\mid \cdots \mid\mid \mathcal{S}_m(x'_m), \tag{2}$$

where $\mathcal{S}_i : \mathbb{F}_2^{s_i} \to \mathbb{F}_2^{s_i}$. We additionally require certain fixed points. If p is of the form $p_{\texttt{gen1}}$ or $p_{\texttt{gen2}}$, then $\mathcal{S}_i(\mathbf{1}^{s_i} = 2^{s_i} - 1) = \mathbf{1}^{s_i}$. If p is of the form $p_{\texttt{gen2}}$, then also $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$.

Almost any invertible AndRX transformation works well for $\mathcal{S}$ and can be implemented in constant time as its components are basic x86 operations. Here we limit ourselves to give some concrete examples for the case $p_{\texttt{gen1}} = 2^n - 1$.

- *Bit Shuffle.* Clearly, both $\mathbf{1}^s$ and $\mathbf{0}^s$ are fixed points under the bit shuffling operation. Moreover, it is essentially for free in hardware.
- *Efficient Linear Operations.* Linear operations over $\mathbb{F}_2^n$ of the form

  $$x \mapsto x \oplus (x \lll i) \oplus (x \lll j)$$

  with non-null $i \neq j$, and where $\lll$ denotes the circular shift operation, are (i) invertible for odd s and (ii) result in $\mathbf{1}^s$ and $\mathbf{0}^s$ being fixed points.
- *Efficient Nonlinear Operations.* Nonlinear operations over $\mathbb{F}_2^n$ such as

  $$x \mapsto x \oplus (\bar{x} \lll 1) \odot (x \lll 2)$$

for odd n, where $\bar{x} := x \oplus \mathbf{1}^s$, are also possible. This corresponds to the χ-function [Dae95, Table A.1] already used in Keccak/SHA-3, which is known to be invertible for $\gcd(n, 2) = 1$. Moreover, $\mathbf{1}^s$ and $\mathbf{0}^s$ are fixed points.

A rotation of the bits at the output of $\mathcal{S}_i$ may be necessary in order to reduce the number of fixed points. Similar examples can be constructed for other primes, as given in Section 4.

Composition $\mathcal{C}$. The final operation $\mathcal{C}$ is the inverse of the decomposition. Given $(x'_1, x'_2, \ldots, x'_m)$ we interpret them as integers and compute

$$y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i-1} s_j} \cdot x'_i \bmod p.$$

One may additionally permute $\{x'_i\}_i$, but our construction does not need this extra operation and we omit it for brevity.

2.2 Well-Definition and Bijectivity

Here we prove that our $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ defined in Eq. (1) and in particular its $\mathcal{S}$ components are invertible and well-defined.

Lemma 1. *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(\mathbf{1}^{s_i}) = \mathbf{1}^{s_i}$, where $i \in \{1, 2, \ldots, m\}$. If $p = 2^{\sum_{i \le m} s_i} - 1 = 2^n - 1$, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection on $\mathbb{F}_p$.*

Proof. Clearly, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection over $\mathbb{Z}_{2^n}$ as it is merely a chunkwise application of invertible S-boxes. Note that $2^n - 1$ is mapped to itself, as $\mathcal{D}$ and $\mathcal{C}$ preserve it by definition, and each S_i maps the binary 1-vector to itself also by definition. Therefore, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ maps $\mathbb{Z}_{2^n-1} = \mathbb{Z}_{2^n} \setminus \{2^n - 1\}$ bijectively to itself. □

Lemma 2. *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$ and $\mathcal{S}_i(\mathbf{1}^{s_i}) = \mathbf{1}^{s_i}$. If $p = 2^n - 2^\eta + 1 = 2^{\sum_{i \le m} s_i} - 2^{\sum_{i \le \ell} s_i} + 1$, then $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection on $\mathbb{F}_p$.*

Proof. Again, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection over $\mathbb{Z}_{2^n}$ as it is merely a chunkwise application of invertible S-boxes. Let us investigate its behaviour on $x \ge 2^n - 2^\eta$.

- If $x = 2^n - 2^\eta$, it is decomposed into $(2^{s_m} - 1, 2^{s_{m-1}} - 1, \ldots, 2^{s_{t+1}} - 1, 0, 0, \ldots, 0)$. All these values are fixed points under $\mathcal{S}_i$, and hence $2^n - 2^\eta$ is mapped to itself by `Bar`.
- If $x > 2^n - 2^\eta$, it is decomposed into $(2^{s_m} - 1, 2^{s_{m-1}} - 1, \ldots, 2^{s_{l+1}-1}, a_l, \ldots, a_2, a_1)$ where at least one of a_i is nonzero. All first $m - t$ values are fixed points, whereas at least one of the last l values is nonzero and thus not mapped to zero. Therefore, x is mapped to some $y > 2^n - 2^\eta$.

Due to the bijectivity of $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ over $\mathbb{Z}_{2^n}$, we obtain that the set $\{x > 2^n - 2^\eta\}$ is mapped to itself and therefore $\mathbb{Z}_{2^n - 2^\eta + 1}$ is mapped to itself as well. □

2.3 `Kintsugi`, Earlier `Bars`, and Side-Channel Considerations

Here, we briefly explain the differences and the analogies between the `Kintsugi` strategy just described and the `Bars` functions proposed in `Reinforced Concrete` (and subsequently used in Tip5). Recall that in `Reinforced Concrete` an element of $\mathbb{F}_p$ is represented as a vector from $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_l}$.

- We rely on the structure of the prime p. Thanks to its composition of a few powers of two, the decomposition now is simply a bit extraction rather than a chain of modular reductions, which is expensive both natively and inside the proof system. The bijectivity of `Kintsugi` is guaranteed under the minor and easily satisfied condition that some specific inputs are fixed points.
- The S-boxes of `Reinforced Concrete` or Tip5 do not have a simple representation, and must be implemented as tables both for native and circuit computations. The `Kintsugi` strategy instantiates the S-boxes with AndRX transformations, which are fast and constant-time in native x86 implementations but can easily be transformed to table lookups for circuits.

Side-Channel Leakage and Countermeasures. Lookup tables in symmetric primitives are a well-known source of side channel leakage. When confidential information is processed (e.g., committing to coin secrets with ZK hash functions in privacy-preserving payment systems), 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 the high-level ideas have found first applications in zero-knowledge proof systems [TBP20]. The lookup-oriented designs `Reinforced Concrete` and Tip5 use specific tables for which a constant-time implementation with reasonable overhead is nontrivial. It is thus of utmost importance to have a design where lookups can be replaced with constant-time operations.

2.4 Statistical and Algebraic Properties

In this section we prove generic statement that link together algebraic and statistical properties of mappings over $\mathbb{F}_p$, which we will use in the security analysis of our construction `Monolith`.

Lemma 3. *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 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 $\mathtt{a}$ we have that $\mathsf{F}(\mathtt{a})$ is the bit representation of $\mathcal{F}(a)$. Then 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.*[6]

Proof. We prove this result by contradiction. Suppose that the degree of F is smaller than d. Then the `XOR` sum of its output over any hypercube of degree d

[6] For example, $\lceil 2^{d-0.5} \rceil$ is odd for $d \in \{2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 21, \dots\}$.

is equal to zero [Lai94], including the hypercube

$$\mathfrak{H} := \{\mathsf{a}_0 = (0,0,\ldots,0),\ldots,\mathsf{a}_{2^d-1} = (0,\ldots,0,\underbrace{1,\ldots,1}_{d \text{ ones}})\}.$$

Note that $\mathcal{F}(a_i) = i^2 < p$ by the definition of d. Now consider $\mathfrak{B} = \{\mathsf{a}_i \in \mathfrak{H} \mid i > 2^{d-0.5}\}$, 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, F does not $\texttt{XOR}$ 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 $\lceil 2^{d-0.5} \rceil$ is odd and $d < \log_2 \sqrt{p}$. □

Lemma 4 (Differential). *Let F be a function that maps $\mathbb{F}_p$ to itself with a differential $\Delta_I \to \Delta_O$ holding with probability $0 < \alpha < 1$, i.e., $|\{x \in \mathbb{F}_p \mid \mathcal{F}(x + \Delta_I) = \mathcal{F}(x) + \Delta_O\}| = p \cdot \alpha$. Then we have*

$$\deg(\mathcal{F}) > \alpha \cdot p\,, \tag{3}$$

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

Proof. By definition, $\mathcal{F}(x + \delta_{in}) = \mathcal{F}(x) + \delta_{out}$ 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 5 (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*

$$\deg(\mathcal{F}) \geq \beta \cdot p\,. \tag{4}$$

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 conclude that F has degree at least equal to $\beta \cdot p$. □

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

Corollary 1. *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*

$$\deg(\mathcal{F}) \geq b\,. \tag{5}$$

3 Feistel Type-3 Layer and the Wide Trail Strategy

Here we introduce another component and explore its security properties. This component, which is the Feistel Type-3 layer [ZMI89], is nonlinear and complements $\texttt{Kintsugi Bars}$. The reason is that even though $\texttt{Bars}$ is a high-degree

function, and thus counteracts algebraic attacks, it also has weak differential properties and is thus vulnerable to differential cryptanalysis. In contrast, this new layer has a low degree but strong resistance against differential attacks. We follow the naming convention of `Reinforced Concrete` (the first look-up based hash function) where the nonlinear layer providing protection against statistical attacks is called `Bricks`, and use the same name for uniformity.

The Feistel Type-3 layer is a member of a larger Feistel family [HR10], which has been largely neglected in the favour of SPN schemes in block cipher and hash function design, primarily for its complexity and worse diffusion properties. However, we have found its simple version with the square mapping particularly attractive as it is cheaper in circuits and, most importantly, its blend with an MDS layer yields differential properties similar to those in regular SPNs.

Concretely, a generalized $\texttt{Bricks}^F$ on t elements $x_1, x_2, \ldots, x_t$ is defined by

$$\texttt{Bricks}^F(x_1, \ldots, x_t) := (x_1, x_2 + \mathcal{F}_1(x_1), x_3 + \mathcal{F}_2(x_2), \ldots, x_t + \mathcal{F}_{t-1}(x_{t-1})), \quad (6)$$

where $\mathcal{F}_i$ are nonlinear functions. While alone it does not provide fast diffusion, a combination with a matrix layer (as suggested in [BMT13; BFM+16]) increases the differential properties. This approach is well-known in the SPN design as the wide trail design strategy [DR01], where one proves a lower bound for the number of "active" nonlinear components in *any* differential trail and thus establishes an upper bound for the success probability of a differential attack. Here we follow this line of research, and for the first time we derive bounds for the SPN structure where the nonlinear layer is a Feistel Type-3 function.[7]

Proposition 1. *Consider an r-round construction, where each round consists of the application of $\texttt{Bricks}^F$ over $\mathbb{F}_q^t$ (for $q = p^s$ with $p \geq 2$ and $s \geq 1$) as in Eq. (6) followed by the multiplication with a $t \times t$ MDS matrix. The minimum number s_{min} of active functions $\mathcal{F}_i$ in any differential trail satisfies*

$$s_{min} \geq (t-1) \cdot \left(\frac{3r - 2 - (-2)^{1-r}}{9} \right) \geq (t-1) \cdot \left(\frac{3r - 2.5}{9} \right).$$

Proof. Denote the number of active words in the input and the output of the i-th $\texttt{Bricks}^F$ layer by a_i and b_i, respectively. Then we exploit two properties.

- Each active input word x_i to $\texttt{Bricks}^F$ activates $\mathcal{F}_i$ if $i < t$, hence a words activate at least $a - 1$ functions $\mathcal{F}_i$.
- Each active output word y_i of $\texttt{Bricks}^F$ implies that $\mathcal{F}_{i-1}$ or $\mathcal{F}_{i-2}$ is active if $i > 1$. Hence b words activate at least $\frac{b-1}{2}$ functions.

Together with the MDS property, which states that $b_k + a_{k+1} \geq t + 1$ for each $k \geq 1$, we obtain the following inequalities for the number s_k of active functions

[7] Our new bound improves the ones recently proposed in [Gra23] for an analogous (but different) scheme.

$\mathcal{F}_i$ in round k:

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

where r is the number of rounds.

Now let $(s_1, s_2, \ldots, s_r)$ be a tuple of values of some valid trail that minimizes $s_{\min} := s_1 + s_2 + s_3 + \ldots + s_r$. Note that this tuple turns all inequalities into strict equations, as otherwise we could reduce $s_{\min}$. 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 to not increase $s_{\min}$. Similarly, if $a_{i+1} < s_{i+1}$, we can increase a_i to make those equal and to not increase $s_{\min}$. Thus we conclude that for an optimal tuple the values b_i and a_{i+1} are the maxima that determine s_i and s_{i+1} respectively. This simplifies our system, i.e.,

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

and even further, i.e.,

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

It is simple to note that if $s_r > 0$, then we could decrease $s_{\min}$. 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 $s_{\min}$ by $s_r \cdot (1 - 1/2 + 1/4 - 1/8 + \cdots) > 0$. 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

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

Substituting these values into the formula for $s_{\min}$, we obtain

$$\begin{aligned}
s_{\min} &= \sum_{i=0}^{r-1} \frac{t-1}{3} \cdot \left(1 + \frac{(-1)^{i+1}}{2^i}\right) = \frac{t-1}{3} \cdot \sum_{i=0}^{r-1} \left(1 + \frac{(-1)^{i+1}}{2^i}\right)\\
&= \frac{t-1}{3} \cdot \left(r - \sum_{i=0}^{r-1} (-2)^{-i}\right) = \frac{t-1}{3} \cdot \left(r - \frac{2 \cdot (1 - (-2)^{-r})}{3}\right)\\
&= (t-1) \cdot \left(\frac{3r - 2 - (-2)^{1-r}}{9}\right).
\end{aligned}$$

□

4 Specification of Monolith

Monolith is a family of permutations which can be used within hash functions and other symmetric constructions. We define the permutation Monolith-64 over $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$ with the state consisting of $t = 8$ or $t = 12$ elements. The permutation Monolith-31 is defined over $p_{\texttt{Mersenne}} = 2^{31} - 1$ with the state consisting of $t = 16$ or $t = 24$ elements.

4.1 Modes of Operation

Monolith supports sponge modes and a 2-to-1 compression function.

Sponge-Based Schemes. First, Monolith can instantiate a sponge [BDP+07; BDP+08] and thus various symmetric constructions such as variable-length hash functions, commitment schemes, authenticated encryption, and stream ciphers. The recently proposed SAFE framework [AKM+22; KBM23] instructs how to handle domain separation and padding in these constructions. In a sponge, 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. As we uniformly suggest the 128-bit security level, we set $c = \left\lfloor \frac{256}{\log_2 p} \right\rfloor$ and $r = 2c$.

2-to-1 Compression Function. We also suggest a fixed-length 2-to-1 *compression function*. Concretely, it takes t $\mathbb{F}_p$ elements as input and produces $t/2$ $\mathbb{F}_p$ elements as output. It is defined as $x \in \mathbb{F}_p^t \mapsto \mathrm{Tr}_{t/2}(\mathcal{P}(x) + x) \in \mathbb{F}_p^t$, where $\mathrm{Tr}_{t/2}$ yields the first $t/2$ elements of the inputs. This compression function can be used in Merkle trees and has recently also been applied in similar constructions, including Anemoi [BBC+23], Griffin [GHR+23], and Poseidon2 [GKS23]. For the 128-bit security level, we set $t = \left\lfloor \frac{512}{\log_2 p} \right\rfloor$, i.e., $t = 8$ for the 64-bit field and $t = 16$ for the 31-bit field (thus factually yielding a little less than 128 bits).

4.2 Permutation Structure

The Monolith permutation is defined as

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

where r is the number of rounds and $\mathcal{R}_i$ over $\mathbb{F}_p^t$ are defined as

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

where Concrete is a linear operation, Bars and Bricks are nonlinear operations over $\mathbb{F}_p^t$, $c^{(1)}, \ldots, c^{(r-1)} \in \mathbb{F}_p^t$ are pseudo-random round constants, and $c^{(r)} = \mathbf{0}$. 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. 3.

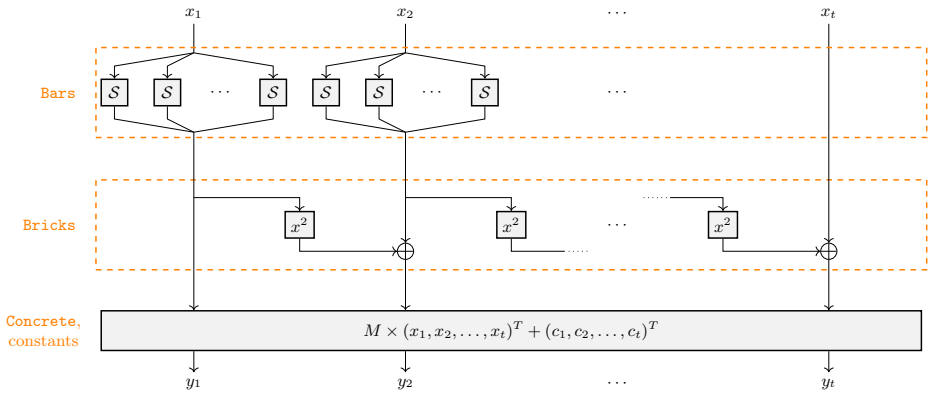

Fig. 3. One round of the `Monolith` construction, where $x_i, y_i \in \mathbb{F}_p$.

4.3 `Bricks`

The component `Bricks` over $\mathbb{F}_p^t$ is defined as a Feistel Type-3 $\texttt{Bricks}^F$ (6) with the square map $x \mapsto x^2$, i.e.,

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

4.4 `Concrete`

The `Concrete` layer 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 circ(23, 8, 13, 10, 7, 6, 21, 8) for $t = 8$ and the matrix 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.[8] 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 native 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 $p_{\texttt{Mersenne}}$.[9] 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 native performance. The problem of finding a fast MDS matrix for this larger state size (which would significantly increase the native performance of `Monolith-31` with $t = 24$) is left as future work.

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

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

4.5 Bars

The Bars layer is defined as

$$\texttt{Bars}(x_1, x_2, \ldots, x_t) := \texttt{Bar}(x_1) \mid\mid \texttt{Bar}(x_2) \mid\mid \cdots \mid\mid \texttt{Bar}(x_u) \mid\mid x_{u+1} \mid\mid \cdots \mid\mid x_t \quad (7)$$

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

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

where $\mathcal{C}, \mathcal{S}$ and $\mathcal{D}$ are the operations defined in Section 2. In the following, we describe them individually for Monolith-64 and Monolith-31.

Bars for Monolith-64. In Eq. (7) 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).

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

$$x = 2^{56}x'_8 + 2^{48}x'_7 + 2^{40}x'_6 + 2^{32}x'_5 + 2^{24}x'_4 + 2^{16}x'_3 + 2^8x'_2 + x'_1.$$

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

S-Boxes $\mathcal{S}$. In Eq. (2) we set $m = 8$. Then all $\mathcal{S}_i$ over $\mathbb{F}_2^8$ are identical as (see [Dae95, Table A.1])

$$\mathcal{S}_i(y) = \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \quad (8)$$

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. (7) 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).

Operations $\mathcal{D}$ and $\mathcal{C}$. The decomposition $\mathcal{D}$ is given by

$$x = 2^{24}x'_4 + 2^{16}x'_3 + 2^8x'_2 + x'_1,$$

where $x'_4 \in \mathbb{Z}_2^7$ and $x'_3, x'_2, x'_1 \in \mathbb{Z}_2^8$. The composition $\mathcal{C}$ is the inverse operation of the decomposition $\mathcal{D}$.

S-Boxes $\mathcal{S}$. In Eq. (2) we set $m = 4$ using $\{8, 7\}$-bit lookup tables. Then the S-boxes are defined as (see [Dae95, Table A.1])

$$\begin{aligned} \forall i \in \{1, 2, \ldots, m-1\}: \quad \mathcal{S}_i(y) &= \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \\ \mathcal{S}_m(y') &= \big(y' \oplus \big((\overline{y'} \lll 1) \odot (y' \lll 2)\big)\big) \lll 1, \end{aligned} \quad (9)$$

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

4.6 Round Constant Generation

The actual values of pseudo-randomly chosen round constants have no impact on the security. For completeness we provide a generation method in Appendix A.3.

4.7 Number of Rounds

In Table 1, we propose to use $r = 6$ rounds for `Monolith`-64 and `Monolith`-31, for which we claim $2\log_2(p_{\texttt{Goldilocks}}) \approx 128$ bits and $4\log_2(p_{\texttt{Mersenne}}) \approx 124$ bits of security, respectively. These numbers are conservatively chosen based on the security analysis proposed in Section 5.

Table 1. Parameters for `Monolith`.

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

5 Security Analysis

As some of the components or combinations are new, our analysis contains several nontrivial ideas and may be of separate interest to cryptanalysts and designers. Here are several insights.

- 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.

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. Since the `Bars` layer is not supposed to have good statistical properties, we simply assume that the attacker can skip it with probability 1.

As the maximum differential probability of a squaring is $1/p$, Proposition 1 immediately implies the following bound.

Corollary 2. *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 characteristics.

Fixed Points. Contrary to `Reinforced Concrete`, the `Bars` layer of `Monolith` has very few fixed points. 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 2 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 Δ_1, Δ_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:

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

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

1. In the inbound phase we arbitrarily choose δ 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 Δ_1 be $\texttt{Concrete}^{-1}(\Delta_2)$.
2. The inbound phase covers the expansion of Δ_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 δ_2, δ_3 such that $\Delta_{8,t} = \Delta_{1,t}$ and then choose δ_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 (δ_3, δ_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

Lower Bound on the Degree over $\mathbb{F}_2$. Using the fact that $\lceil 2^{d-0.5} \rceil$ is odd for $d = 15$ and $d = 30$, Lemma 3 implies the following bound on the degree over $\mathbb{F}_2$.

Proposition 2. *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 (i) $d = 30$ for $p = 2^{64} - 2^{32} + 1$, and (ii) $d = 15$ 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 6. *Let $n > 4$.*

- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (8)*

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

  is at least 13/32.
- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (9)*

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

  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 if the product in the S-box bit mapping is 0 whenever bit i is `XOR`ed or is part of the product, i.e.,

$$\overline{y_{i+1}} \odot y_{i+2} \odot y_{i+3} = 0, \qquad y_{i+1} \odot y_{i+2} = 0,$$
$$\overline{y_{i-1}} \odot y_{i+1} = 0, \qquad \overline{y_{i-2}} \odot y_{i-1} = 0.$$

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. (9) we observe that a single bit difference in bit i is mapped to a single bit difference in bit $i-1$ if

$$\overline{y_{i+1}} \odot y_{i+2} = 0, \quad y_{i+1} = 0, \quad \overline{y_{i-1}} = 0,$$

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

Lemma 7. *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 6 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 6 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 3. *The* `Bars` *operation (and its inverse) has degree at least (i)* 2^{62} *for* $p = 2^{64} - 2^{32} + 1$, *and (ii)* 2^{29} *for* $p = 2^{31} - 1$.

The real degree and density values for smaller p are presented in Appendix.

5.4 The CICO Problem for Keyless Algebraic Attacks

A large class of attacks on permutation-based hash functions is reduced to the CICO problem [BDP+09].

Definition 1 (CICO Problem). *A permutation* $\mathcal{P} : \mathbb{F}_p^t \to \mathbb{F}_p^t$ *is secure 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* $\mathcal{P}(I_1 \,||\, \underbrace{\mathbf{0}}_{v \text{ words}}) = \underbrace{\mathbf{0}}_{v \text{ words}} \,||\, O_2$.

We use to argue the security of `Monolith` against some classes of algebraic attacks (in particular Gröbner basis ones) as follows. First, we 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 situations.

Univariate Case. A univariate system appears if $v = t-1$ or we guess $t-v+1$ variables. Note that our guess may be invalid if the number of equations exceeds the number of variables, so we have to repeat the guess p^{v-1} times.

- If $v = 1$ and we have guessed $t-2$ variables, then 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 $v > 1$, and we have guessed $t-v-1$ variables, then the chance of a CICO solution for a guess is p^{v-1}. A system of polynomial equations has 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 obtain the value of a last variable. Therefore the total complexity still exceeds $p \cdot p^{v-1} = p^v$.

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. We argue that even the first step is prohibitively expensive for `Monolith`.

The complexity of computing a *degrevlex* Gröbner basis, which we denote as $\mathcal{C}_{\text{GB}}$, is difficult to estimate for structured primitives like `Monolith`. For regular sequences with m equations of degree $d_1, d_2, \ldots, d_m$ and m variables it is shown to be

$$\mathcal{C}_{\text{GB}}^{regular} = \mathcal{O}\left(\binom{m + d_{\text{th}}}{m}^{\omega}\right) \tag{10}$$

where d_{th} is called the (theoretical) degree of regularity and computed as $1 + \sum_{i=1}^{m}(d_i - 1)$, and ω, the linear algebra exponent, for equations of reasonable size is close to 2.8.

Unfortunately for real usecases the complexity of GB computation is known to be quite volatile [BGL20; Sau21]. The main source of discrepancy is arguably the degree of regularity, which is practically the maximum degree reached by polynomials in the GB algorithms. Another problem is scalability: it is nontrivial to scale down an original system of equations to some variant that is solvable on a PC, and get estimates from there. We tackle these problems and the full algebraic security of `Monolith` as follows:

– We consider a very small, weakened version of `Monolith`, denoted `Monolith`-Weak1R – with a small prime p, a state with $t = 4$ elements, only one round $\mathcal{F} := \texttt{Concrete} \circ \texttt{Bricks} \circ \texttt{Bars} \circ \texttt{Concrete}$, and only two `Bar` instances in the round.
– We suggest an arguably optimal representation of `Monolith`-Weak1R as a system of polynomial equations of small degree.
– For various small primes we run an actual GB computation and show that the experimental degree of regularity, d_{mag}, is lower bounded by $d_{\text{th}}/4$.
– We argue that the actual GB computation cost in our experiments can be lower bounded by a modified version of (10) with a high security margin:

$$\mathcal{C}_{\text{GB}}^{\texttt{Monolith}-1R} \gg \mathcal{C}^{bound}(n, d_{\text{th}}) = \binom{n + d_{\text{th}}/4}{n} \tag{11}$$

where n is uniformly 10 in our model and D depends on p and the S-box structure. We note, we use $\omega = 1$ in $\mathcal{C}^{bound}(n, d_{\text{th}})$.
– We show that the application of (10) to the full-size state of `Monolith` and the original primes yields a complexity estimate of 2^{334} and higher.

Based on that, we argue that the full version of `Monolith` is secure against a GB attack.

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 $2^{s_1}, 2^{s_2}, \ldots, 2^{s_m}$:

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

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

$$\mathcal{L}_i(x_i) := \sum_{0 \le k \le 2^{s_i}-1} \mathcal{S}_i(k) \prod_{\substack{0 \le j \le 2^{s_i}-1 \\ j \ne k}} \frac{x_i - j}{k - j}.$$

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

Algebraic Model for One Round of `Monolith`-Weak1R. We consider a CICO problem with $t = 4$ words, i.e., we are looking for $x_2, x_3, x_4 \in \mathbb{F}_p$ and $y_2, y_3, y_4 \in \mathbb{F}_p$ such that

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

Table 2. Results of Gröbner basis computations on small-scale instances of a single round of Monolith-Weak1R in the CICO setting. d_{mag} denotes the maximum degree reached during a GB computation with Magma. T is time in microseconds (10^{-6}). Extrapolated estimates are in italic.

	Monolith-Weak1R				Monolith-1R	
p	13	29	61	113	31-bit	64-bit
m, n	10, 10	10, 10	10, 10	10, 10	64, 64	48, 48
s_i	2, 2	2, 3	2, 4	4, 3	8,8,8,7	8,...,8
d_{th}	18	34	66	74	9177	9181
d_{mag}	11	14	19	24	*2295*	*2296*
$d_{\text{th}} : d_{\text{mag}}$	1.62	2.43	3.47	3.08	*4*	*4*
$\log_2 T$	16.5	21.5	25.5	30.5		
$\log_2 \mathcal{C}^{bound}(n, d_{\text{th}})$	10.8	16	23	24.3	419.8	333.7

where Monolith-Weak1R's function $\mathcal{F} \coloneqq \texttt{Concrete} \circ \texttt{Bricks} \circ \texttt{Bars} \circ \texttt{Concrete}$ is 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 use the following system of equations:

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

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.

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 2. 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 see that the real degree d_{mag} is always higher than $d_{\text{reg}}/4$, and we also see that $\mathcal{C}^{bound}$ values are indeed a safe lower bound for the actual computation time for instances of Monolith-Weak1R. Assuming the $\mathcal{C}^{bound}$ lower bound, and

taking into account that Monolith-1R is stronger than Monolith-Weak1R, we obtain that the Gröbner basis cost for the Monolith-1R CICO problem should be at least 2^{420} for 31-bit and 2^{333} for the 64-bit version. This hints that the full Monolith is secure against Gröbner basis attacks. We conclude that using Monolith with 6 rounds provides ample security margin against Gröbner basis attacks.

6 Performance Evaluation

6.1 Native Performance

In this section we compare the native performance of Monolith to its competitors with results in Table 3. All benchmarks were taken on an AMD Ryzen 9 7900X CPU (singlethreaded, 4.7 GHz).

We included implementations of Monolith into the framework in [IAI21], and also added instantiations of widely popular Poseidon [GKR+21], its modification Poseidon2 [GKS23], and also Griffin [GHR+23] with $p = 2^{64} - 2^{32} + 1$ following their original instance generation scripts.[10] [11] We benchmark these hash functions with a state size of $t = 8$ for the compression mode and of $t = 12$ for the sponge mode in order to have a fair comparison. We also compare against Tip5 with its fixed state size of $t = 16$ using the implementation from [SLS+23],[12] and against Tip4′, a faster instance of Tip5 with a fixed state size $t = 12$, using the implementation from [Sal23].[13] We also compare against Reinforced Concrete instantiated with the scalar field of the BN254 curve, and against SHA3-256/SHA-256 as implemented in RustCrypto.[14]

Finally, we compare Monolith-31 with Poseidon and Poseidon2 over the $p_{\texttt{Mersenne}}$ prime field and state sizes of $t = 16$ and $t = 24$ (again for sponge and compression mode), as well as for a constant time implementation (constant time $\mathbb{F}_p$ operations and no lookup tables).

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 native performance with 210 ns for $t = 16$. This is significantly

[10] The source code is available at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/tree/master/plain_impls.
[11] See, e.g., https://github.com/anemoi-hash/hash_f64_benchmarks
[12] https://github.com/Neptune-Crypto/twenty-first
[13] https://github.com/Nashtare/winterfell
[14] https://github.com/RustCrypto/hashes

Table 3. Native performance comparison in nano seconds (ns) of different hash functions for variable and constant time implementations. Benchmarks are given for one permutation call, i.e., hashing ≈ 500 bits for all but SHA functions.

Hashing algorithm	Time (ns)		Const. Time (ns)	
	2-to-1	sponge	2-to-1	sponge
$p = 2^{64} - 2^{32} + 1$:	$t = 8$	$t = 12$	$t = 8$	$t = 12$
`Monolith`-64	**129.9**	**210.5**	**148.5**	**230.4**
Poseidon	1897.6	3288.7	2347.6	4059.1
Poseidon2	944.6	1291.5	1149.2	1617.9
Rescue-Prime	12128.0	19095.0		
Griffin	1815.0	1988.4		
Tip5 ($t = 16$)		463.6		
Tip4′		247.9		
$p = 2^{31} - 1$:	$t = 16$	$t = 24$	$t = 16$	$t = 24$
`Monolith`-31	**210.3**	**1015.3**	**237.9**	**1120.5**
Poseidon	4478.8	8539.7	4372.9	8538.0
Poseidon2	792.8	1257.4	840.7	1355.3
Other:				
`Reinforced Concrete` (BN254)		1467.1		
SHA3-256				189.8
SHA-256			45.3	

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 native 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.

Another advantage of `Monolith` over Tip5, Tip4′, and `Reinforced Concrete` is that its native performance does not rely on lookup tables and its structure allows for constant-time implementations without significant performance loss. The binary χ-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].

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

Table 4. Plonkish arithmetization comparison for various 64-bit schemes. The numbers are for a single permutation.

Primitive	Lookups	Nonlinear constraints	Degree	Witness size	Area-degree product
`Monolith`-64-compression	192	44	**2**	460	920
`Monolith`-64-sponge	192	64	**2**	480	960
Tip5	160	60	7	380	2660
Tip4′	160	40	7	360	2520
Poseidon/Poseidon2 (sponge)	0	118	7	118	826
Rescue-Prime (sponge)	0	96	7	96	**672**

Table 5. Performance of proving a Poseidon and `Monolith` permutation using $p_{\texttt{Goldilocks}}$ and sponge mode in the Plonky2 proof system.

Primitive	Prove Time ms	Verify Time ms	Proof Size B
`Monolith`-64-sponge	3.49	0.63	112732
Poseidon	6.23	1.12	70288

`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 completeness, we give the runtime of each part of the `Monolith` permutation for both a constant- and variable-time version in Appendix C.

6.2 Performance in Proof Systems

A modern zero-knowledge proof system defines *arithmetization rules* for the circuit it attempts to prove. Most new proof systems support the *Plonkish* arithmetization, where all input, output, and intermediate variables are placed into a *witness matrix* W with m columns and n rows. The data in each row is restricted by polynomial equations determining the values and computations being used. One of these generic equations of degree 2 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 [GWC19]. 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 set of values in a predefined table $\mathfrak{T}$.

A precise comparison of different arithmetizations is hard without implementing and testing. However, a significant part of the work is to construct m degree-n polynomials for the witness columns and to prove that they satisfy the polynomial equations. The total work is then estimated as an element in $\mathcal{O}(d \cdot n \cdot m)$, where d is the maximum degree of a row polynomial. The cost of using table lookups for FRI-based schemes is currently equivalent to the use of a single polynomial of degree $t = \max\{n, |\mathfrak{T}|\}$.

In this section we give possible arithmetizations for translating `Monolith` into a set of Plonkish constraints and refer to Appendix D.1 for 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 Arithmetization. Each composition $\texttt{Concrete} \circ \texttt{Bricks}$ is described with t polynomial equations of degree 2. Then, for each `Bar` in the `Bars` layer, we enforce the correct relations with $x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i$ and $y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i$, while also making sure that the limbs in the decomposition correspond to field elements. For $p_{\texttt{Goldilocks}}$, this means enforcing that either the least significant 32 bits of `Bar`'s input are 0 or the most significant bits are not all 1, i.e.,

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

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

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

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 also include the range checks for each input which are also necessary for the correctness of the constraints above.

Apart from $2m$ lookup variables per `Bar`, we define u variables at the output of the first `Concrete` layer (these are the inputs to the `Bars` layer) and t variables at the output of each of the following `Concrete` layers (except for the last one). The reason is that the variables after the first `Concrete` layer store linear relations in the input, and only the u variables entering the `Bars` layer are needed. For the last layer, the output variables can be used directly. In total, we have $6 \cdot (2um + u) + 5t + u$ variables, where $\{u = 4, m = 8\}$ for the $p_{\texttt{Goldilocks}}$ case and $\{u = 8, m = 4\}$ for the $p_{\texttt{Mersenne}}$ case (considering S-boxes of ≈ 8 bits).

In Table 4 we compare the (non-optimized) arithmetization of `Monolith` with the ones of other 64-bit designs (see Appendix D.2 for details). 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. Poseidon, Poseidon2, and *Rescue*-Prime due to their comparably small witness size and no lookup tables are estimated to still provide faster proving performance, closely followed by `Monolith`-64 with its low-degree nonlinear layers. Again, we stress that these numbers are derived from non-optimized arithmetizations and are subject to change. For example, one can leverage the low degree of `Monolith` to reduce witness size by trading with a larger degree round function. We refer to Appendix D.3 for details.

Furthermore, these estimates are based on a simplified performance metric (are-degree-product) which does not consider every aspect of prover performance and benchmarks in real proof systems might differ.

Benchmarks in Plonky2. We implemented Monolith-64 in the Plonky2[15] proof system to verify the estimations of Table 4.[16] Since Plonky2 already comes with a custom gate of Poseidon in sponge mode ($t = 12$) using $p_{\texttt{Goldilocks}}$ where the whole gate is put into just one row of the trace, we implement Monolith-64-sponge with the same parameters. To highlight the main advantage of Monolith-64, namely its fast native performance, we benchmark proving a Monolith-64 permutation while using Monolith-64 as the hash function to build the Merkle trees in Plonky2. Similarly, we benchmark Poseidon when using Poseidon as the Plonky2 hash function (which is the default setting in Plonky2). The results can be seen in Table 5. One can observe that since Monolith requires more witnesses than Poseidon and both gates use just one row in the trace, the resulting proof is larger. However, the combination of proving Monolith-64 while using it at as the Plonky2 hash function leads to half the prover and verifier runtime compared to Poseidon.

Acknowledgments

This work was partially supported by a gift from the Ethereum foundation. 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 was 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.

Finally, we thank Nicholas Mainardi for helping with the implementation of Monolith in Plonky2 and for improving the efficiency of the gate.

References

[22a] *ZKEVM Introduction*. https://github.com/privacy-scaling-explorations/zkevm-specs/blob/master/specs/introduction.md. 2022 (cit. on p. 2).

[22b] *Polygon zkEVM Documentation*. https://docs.hermez.io/zkEVM/Overview/Overview/. 2022 (cit. on p. 2).

[15] https://github.com/mir-protocol/plonky2

[16] Our implementation is available at https://github.com/HorizenLabs/monolith.

[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), pp. 1–45 (cit. on p. 2).

[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. 2016, pp. 191–219 (cit. on p. 2).

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. "NORX: Parallel and Scalable AEAD". In: *ESORICS 2014*. Vol. 8713. LNCS. 2014, pp. 19–36 (cit. on p. 6).

[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. 13).

[ANW+13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. "BLAKE2: Simpler, Smaller, Fast as MD5". In: *ACNS 2013*. Vol. 7954. LNCS. 2013, pp. 119–135 (cit. on p. 2).

[Bal23] Balazs Komuves. *hash-circuits*. 2023. URL: https://github.com/bkomuves/hash-circuits (visited on 10/06/2023) (cit. on p. 5).

[BBC+23] 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". In: *CRYPTO 2023*. Vol. 14083. LNCS. 2023, pp. 507–539 (cit. on pp. 2, 13).

[BBH+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *CRYPTO 2019*. Vol. 11694. LNCS. 2019, pp. 701–732 (cit. on pp. 2, 3).

[BC23] Benedikt Bünz and Binyi Chen. "ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols". In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/620, p. 620 (cit. on pp. 1, 2).

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. "Higher-Order Differential Properties of Keccak and *Luffa*". In: *FSE 2011*. Vol. 6733. LNCS. 2011, pp. 252–269 (cit. on pp. 38, 39).

[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. 2020, pp. 299–328 (cit. on p. 38).

[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. 13).

[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. 2008, pp. 181–197 (cit. on p. 13).

[BDP+09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak sponge function family main document". In: *Submission to NIST (Round 2)* 3.30 (2009), pp. 320–337 (cit. on p. 20).

[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. 39).

[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 2018*. Vol. 10892. LNCS. 2018, pp. 400–418 (cit. on pp. 2, 39).

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. Avaiable at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf. 2005 (cit. on p. 9).

[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), pp. 2074–2089 (cit. on pp. 4, 11).

[BGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. *STARK Friendly Hash – Survey and Recommendation*. Cryptology ePrint Archive, Paper 2020/948. https://eprint.iacr.org/2020/948. 2020 (cit. on p. 21).

[BMT13] Thierry P. Berger, Marine Minier, and Gaël Thomas. "Extended Generalized Feistel Networks Using Matrix Representation". In: *SAC 2013*. Vol. 8282. LNCS. 2013, pp. 289–305 (cit. on pp. 4, 11).

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *CRYPTO 1990*. Vol. 537. LNCS. 1990, pp. 2–21 (cit. on p. 17).

[But22] Vitalik Buterin. *What we want out of STARK signature aggregation*. available at https://t.ly/UZMKw. 2022 (cit. on p. 2).

[CFG+22] Shumo Chu, Boyuan Feng, Brandon H. Gomes, Francisco Hernández Iglesias, and Todd Norton. *MantaPay Protocol Specification*. available at https://github.com/Manta-Network/spec/blob/main/manta-pay/spec.pdf. 2022 (cit. on p. 2).

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In:

EUROCRYPT 2020. Vol. 12105. LNCS. 2020, pp. 769–793 (cit. on pp. 2, 3).

[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. 4, 8, 15).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *FSE 1997*. Vol. 1267. LNCS. 1997, pp. 149–165 (cit. on p. 38).

[DR01] Joan Daemen and Vincent Rijmen. "The Wide Trail Design Strategy". In: *Cryptography and Coding - IMA International Conference 2001*. Vol. 2260. LNCS. 2001, pp. 222–238 (cit. on p. 11).

[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 pp. 4, 16).

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. "cq: Cached quotients for fast lookups". In: *IACR Cryptol. ePrint Arch.* (2022). https://eprint.iacr.org/2022/1763 (cit. on p. 3).

[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. 21).

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications". In: *CRYPTO 2023*. Vol. 14083. LNCS. 2023, pp. 573–606 (cit. on pp. 2, 13, 24, 39).

[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*. 2022, pp. 1323–1335 (cit. on pp. 3, 6).

[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. 2, 4, 24, 39).

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. "Poseidon2: A Faster Version of the Poseidon Hash Function". In: *AFRICACRYPT 2023*. Vol. 14064. LNCS. 2023, pp. 177–203 (cit. on pp. 3, 13, 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. 39).

[Gra23] Lorenzo Grassi. "Bounded Surjective Quadratic Functions over Fnp for MPC-/ZK-/FHE-Friendly Symmetric Primitives". In: *IACR Trans. Symmetric Cryptol.* 2023.2 (2023), pp. 94–131 (cit. on p. 11).

[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. 1, 26).

[Hab23] Ulrich Haböck. *Brakedown's expander code*. Cryptology ePrint Archive, Paper 2023/769. https://eprint.iacr.org/2023/769. 2023 (cit. on p. 3).

[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. 3).

[HR10] Viet Tung Hoang and Phillip Rogaway. "On generalized Feistel networks". In: *Annual Cryptology Conference*. Springer. 2010, pp. 613–630 (cit. on p. 11).

[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).

[KBM23] Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink. "Generic Security of the SAFE API and Its Applications". In: *IACR Cryptol. ePrint Arch.* (2023). to appear at ASIACRYPT'23, p. 520 (cit. on p. 13).

[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. 2).

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE 1994*. Vol. 1008. LNCS. 1994, pp. 196–211 (cit. on pp. 17, 38).

[KS23] Abhiram Kothapalli and Srinath T. V. Setty. "HyperNova: Recursive arguments for customizable constraint systems". In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/573, p. 573 (cit. on p. 2).

[KST22] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *CRYPTO 2022*. Vol. 13510. LNCS. 2022, pp. 359–388 (cit. on pp. 1, 2).

[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. 10).

[MRS+09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. "The Rebound Attack: Cryptanalysis of Reduced Whirlpool

and Grøstl". In: *FSE 2009*. Vol. 5665. LNCS. 2009, pp. 260–276 (cit. on p. 18).

[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 p. 39).

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *CT-RSA 2006*. Vol. 3860. LNCS. 2006, pp. 1–20 (cit. on p. 9).

[Pag02] D. Page. *Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel*. Cryptology ePrint Archive. https://eprint.iacr.org/2002/169. 2002 (cit. on p. 9).

[PH23] Shahar Papini and Ulrich Haböck. "Improving logarithmic derivative lookups using GKR". In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/1284 (cit. on p. 3).

[Pol22] Polygon. *Introducing Plonky2*. 2022 (cit. on pp. 3, 6).

[Pol23] Polygon. *Plonky3*. 2023. URL: https://github.com/Plonky3/Plonky3 (visited on 06/12/2023) (cit. on p. 3).

[PSS19] Alexey Pertsev, Roman Semenov, and Roman Storm. *Tornado Cash Privacy Solution Version 1.4*. available at https://t.ly/ys_pW. 2019 (cit. on p. 2).

[RIS23a] Jeremy Bruestle (RISC0). private communication. 2023 (cit. on p. 3).

[RIS23b] RISC Zero. *RISC Zero : General-Purpose Verifiable Computing*. 2023 (cit. on pp. 3, 5).

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. *Rescue-Prime: a Standard Specification (SoK)*. Cryptology ePrint Archive, Report 2020/1143. 2020 (cit. on p. 2).

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

[Sau21] Jan Ferdinand Sauer. *Blog: Gröbner Basis – Attacking a Tiny Sponge*. available at https://jfs.sh/blog/gb-attacking-tiny-sponge/. 2021 (cit. on p. 21).

[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, 6, 14, 24).

[STW23] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. "Unlocking the lookup singularity with Lasso". In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/1216 (cit. on p. 3).

[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. 9, 25).

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *TCC 2008*. Vol. 4948. LNCS. 2008, pp. 1–18 (cit. on p. 2).

[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. 14).

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. "Caulk: Lookup Arguments in Sublinear Time". In: *CCS*. ACM, 2022, pp. 3121–3134 (cit. on p. 3).

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive. https://eprint.iacr.org/2022/1565. 2022 (cit. on pp. 1, 3).

[Zha22] Ye Zhang. *Introducing zkEVM*. https://scroll.io/blog/zkEVM. 2022 (cit. on p. 2).

[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. 4, 10).

Table of Contents

SUPPLEMENTARY MATERIAL

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 ρ-bit inputs result in an output of at most 2ρ 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.

A.3 Generation of Round Constants

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 $t = 16, r = 6$.

B Security Analysis – Additional Material

B.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 η 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 6. 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 6, where always more than 99% monomials are reached together with a degree close to the maximum.

Table 6. 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)

Table 7. 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 7.

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.

B.2 Non-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+23], we ignore zero-sum partitions for practical applications.

C Benchmarks of Different Round Functions

In Table 8, we give the runtime of each part of the `Monolith` permutation for both a constant- and variable-time implementation.

Table 8. Native performance of each different round function in `Monolith`. Implemented in Rust. * 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*	31.9	138.1*
`Bricks`	17.0	21.7	17.0	21.7
`Bars`	8.4	12.0	8.4	12.0

D Arithmetization Details

D.1 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. First, 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. Finally, the Concrete layer can be included in the constraints of Bricks and Bars, resulting in a total for $R \cdot (1027 + t)$ R1CS constraints for Monolith-64 and $R \cdot (943 + t)$ constraints for Monolith-31, where R is the number of rounds.

D.2 Circuits for Other Hash Functions

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 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 outputs of the S-boxes, and link the others via linear equations.

Regarding *Rescue*-Prime, an instance with $t = 12$ defined for $p_{\texttt{Goldilocks}}$ requires 8 rounds which each consist of two subrounds which alternate between nonlinear layers featuring the x^d and $x^{1/d}$ power maps. Due to this construction one can find degree-7 constraints spanning a whole round of rescue, leading to 96 degree-7 constraints in total.

D.3 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 $w_1^{(3)}, \ldots, w_4^{(3)}$ 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 $w_5^{(1)}, \ldots w_{12}^{(1)}, w_1^{(2)}, \ldots, w_4^{(2)}$ of degree 2 and by the 4 new variables $w_1^{(4)}, \ldots, w_4^{(4)}$ resulting from the table lookups. After applying the next `Bricks` and `Concrete` layers, we arrive at a state defined by 12 polynomial constraints in $w_5^{(1)}, \ldots w_{12}^{(1)}, w_1^{(2)}, \ldots, w_4^{(2)}, w_1^{(4)}, \ldots, w_4^{(4)}$ of degree 4. A graphical overview of this approach is shown in Fig. 4.

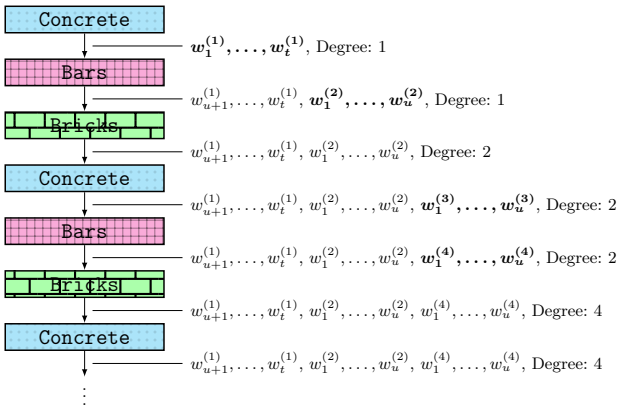

Fig. 4. Variables (or trace elements) when using `Monolith` with degree-4 constraints. Newly added variables are emphasized in **bold** and the degree indicates the maximum degree of the polynomial equations describing the corresponding state in the given variables.

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