The XHash Family for ZK-Friendly Hash Functions

Tomer Ashur[1], Amit Singh Bhati[1,4], Al Kindi[2], Mohammad Mahzoun[1], Léo Perrin[3], Sundas Tariq[1,4]

[1]3MI Labs, Leuven, Belgium.
[2]Polygon.
[3]Inria, Paris, France.
[4]COSIC, KU Leuven, Leuven, Belgium.

Contributing authors: tomer@3milabs.tech; amitsingh.bhati@3milabs.tech; al-kindi-0@protonmail.com; mairon.mahzoun@3milabs.tech; perrin.leo@gmail.com; sundas.tariq@3milabs.tech;

Abstract

Zero-knowledge proofs are widely used in real-world applications, such as authentication, access control, blockchains, and cryptocurrencies. The underlying hash function is a core element in these zero-knowledge proof systems, and it plays a vital role in its efficiency. While traditional hash functions, such as SHA3 or BLAKE, are efficient on CPU architectures, they perform poorly within proof systems. This is primarily because these systems require functions that operate efficiently over finite fields, without mixing operations from different algebras. To address this challenge, a new paradigm called Arithmetization-Orientation has emerged. These designs are tailored for efficiency within proof systems while providing reliable security guarantees. Several such hash functions have been proposed, many have been successfully targeted by algebraic attacks or fail to provide optimal performances.

In this work, we provide a new framework to design such hash functions that prevent these attacks and allow a highly efficient implementation in various proof systems. To this end, we combine the Marvellous design strategy with a new type of S-boxes and propose a new security argument against algebraic attacks that relies on a single well-defined and reasonable conjecture of a novel type.

Using these techniques in the framework, we construct four new hash functions, XHash12-Goldilocks, XHash8-Goldilocks, XHash24-M31, and XHash16-M31. These high-performance hash functions are designed for ZK-STARKs and Circle-STARKs.

Keywords: Zero-knowledge, Arithmetization Oriented, Hash Functions, XHash, Goldilocks prime, Mersenne prime, Circle STARKs, Algebraic Attacks

1 Introduction

Zero-knowledge (ZK) proof systems are advanced cryptographic protocols used to prove the validity of a statement to a party (a verifier) without revealing any further information. ZK systems are used in various applications, such as banking systems and blockchains. A pioneer ZK proof system is ZK-STARK [1], which is a scalable proof system widely used in industrial applications such as Polygon Miden and Starknet. A core building block in a ZK-STARK is its underlying hash function. The pursuit of efficient ZK-STARKs has motivated the need for Arithmetization-Oriented (AO) primitives, *i.e.,* cryptographic primitives with low arithmetic complexity. Examples of such AO designs are [2–18].

1.1 AO Hash Functions

The Marvellous design strategy [17] introduced the Rescue hash function, a major advancement in the domain of AO primitives by introducing non-procedural computations. While Rescue is competitive inside the STARK, its security-first approach leaves room for improvement when executed on more standard platforms such as a CPU or an FPGA. Indeed, subsequent improvements were obtained in follow-up works such as Rescue-Prime (RP) [19] and, more recently, Rescue-Prime Optimised (RPO) [20] where Ashur, Kindi, Meier, Szepieniec, and Threadbare published an optimised variant for the specific case of 2-to-1 compression of elements over the Goldilocks prime field $\mathbb{F}_G$, with $G = 2^{64} - 2^{32} + 1$ for 128 and 160 bits of security. RPO offers the same performance as Rescue-Prime when evaluated inside the STARK but is about 40% faster on a standard CPU. Finding a particularly efficient MDS matrix achieved the lion's share of this improvement without changing the overall design or introducing new operations.

In parallel to these developments, other AO algorithms were proposed, such as Griffin [3], Anemoi [4], and Arion [21]. While their performances were promising, a new type of algebraic attack was recently proposed [22], raising concerns about their security. In particular, some of the initial security assumptions were shown to be optimistic, as the attack step whose complexity had been lower-bounded during the design phase could be bypassed. Consequently, certain versions of Griffin, initially designed to provide 128 bit security, were later estimated to achieve around 70 bits. The Marvellous family, however, remains unaffected at this stage.

1.2 Improved Design Techniques

This work presents a framework that combines the resilience of the SPN-based Marvellous approach with the good performance of more recent designs, introducing two novel concepts for constructing such algorithms securely and efficiently.

The first is exponentiation over an extension field. Arithmetic over an extension field is quite common in the design of traditional symmetric-key algorithms such as AES [23], the base field being $\mathbb{F}_2$ and the extension field $\mathbb{F}_2^8$. A similar method was used for some binary AO primitives, namely Vision [17] and Chaghri [2]. However, relying on field extensions to increase security was never considered for primitives operating over large prime fields.

The second is a new type of security argument against algebraic attacks. Such a tool is sorely needed: until now, designers have argued the security of their primitives by estimating the time complexity of one of the attack steps (either the computation of the Gröbner basis or its change of order). However, the authors of [22] showed that algebraic attacks could be run completely differently: their approach (the *FreeLunch* monomial order approach) gives the Gröbner basis at no cost, and the *FreeLunch* system they obtain can be solved using a tailored algorithm that is more efficient than usual change of order algorithms. Until now, no design strategy is known to prevent attacks of this type. However, we formulate a reasonable conjecture under which it becomes possible to lower bound a crucial quantity, the *ideal degree*, that quantifies the complexity of an algebraic system.

1.3 Our Contributions

In this paper, we introduce the tools described above in greater detail, namely the S-boxes operating over the extension field and the security arguments against *FreeLunch*-type attacks. Putting it all together, we design a new permutation family called XHash: it reuses elements from the Marvellous designs and the new wider S-boxes so that we can implement our new security arguments. Concretely, we propose four new permutations:

1. XHash12 and XHash24: interleaving between RPO rounds and a new type of round.
2. XHash8 and XHash16: are more aggressively optimised variants of XHash12 and XHash24 respectively, by not applying the expensive $x^{\frac{1}{\alpha}}$ S-box to every state element.

Therefore, we propose two secure and efficient hash functions in the Circle STARKs setting based on the permutations XHash24 and XHash16 named XHash24-M31 and XHash16-M31. Moreover, we propose two secure and efficient hash functions in the traditional STARKs setting based on the permutations XHash12 and XHash8 named XHash12-Goldilocks and XHash8-Goldilocks.

Overview of the Paper. In Section 2, we introduce the necessary notation and required background for following the paper. In Section 3, we describe new approaches to design AO primitives. In Section 4, we discuss the design rationale of the XHash family and propose the specifications of new hash functions. In Section 5, we analyse the security of XHash8-Goldilocks and XHash16-M31 against statistical and algebraic attacks. In Section 6, we present the benchmark results. Finally, we conclude the work in Section 7.

2 Preliminaries

In this section, we introduce some basic definitions as well as the specifications of Rescue-Prime Optimised, using it as a starting point for our designs.

2.1 Finite Field Algebra

Let $\mathbb{F}_p$ be a finite field of order p. An extension field is a quotient ring, *i.e.,* $\mathbb{F}_{p^n} = \mathbb{Z}_p[x]/\langle f(x)\rangle$ where $f(x) = \sum_{i=0}^{n} a_i x^i \in F[x]$ is an n-degree irreducible polynomial with $a_i \in \mathbb{F}_p$ and $n \geq 1$.

We say that $\alpha \in \mathbb{F}_p$ is a primitive element (or a generator) of $\mathbb{F}_p$ when the set $\{0, \alpha, \alpha^2, \ldots, \alpha^{p-1}\}$ contains all elements in the field. The order of any nonzero element $\beta \in \mathbb{F}_p$ is the smallest positive integer k such that $\beta^k = 1$. We denote the order of an element by $|\beta| = k$.

2.2 Notation

In this work, we use two finite fields: the Goldilocks prime field $\mathbb{F}_G$, where $G = 2^{64} - 2^{32} + 1$, and a Mersenne prime field (M31) $\mathbb{F}_M$, with $M = 2^{31} - 1$. Their cubic extension fields, $\mathbb{F}_{G^3}$ and $\mathbb{F}_{M^3}$ will also be considered.

Vectors and matrices are denoted by capital Latin letters. For example, the vector S of size m is denoted by $S = (S[0], \ldots, S[m-1])$ and the elements of a matrix M with dimensions $m \times m$ are denoted by $M[i][j]$ where $0 \leq i, j < m$. We use $\boxplus$ to denote addition over prime fields. Finally, ℓ denotes the number of inverse power maps applied in each round.

2.3 Roots of Unity

We say that a complex number $\gamma \in \mathbb{C}$ is an nth root of unity if $\gamma^n = 1$. We denote by U_n the set of all n^{th} roots of unity and use Euler's formula to find it; *i.e.,*

$$U_n = \{e^{\frac{2k\pi i}{n}} \mid k \in \{1, \ldots, n\}\}.$$

In the sequel, we will use a generator $\alpha_{U_n} \in \mathbb{C}$ of U_n to construct an $m \times m$ MDS matrix in M31. To do so, we need to find the quadratic residues of different elements in M31 corresponding to α_{U_n}.

Let $x, y \in \mathbb{F}_p$, we say that y is a quadratic residue of x if and only if $x^2 = y$. For the convenience of readers, we give an example for the calculation of the 16th primitive root of unity and quadratic residues in Appendix A.

2.4 Sponge Construction

The sponge construction [24] is a generic method for constructing a hash function from a fixed-length public permutation. The original sponge construction works with permutations over $\mathbb{F}_2$, but the concept of sponges and its security over other finite fields is described in [25]. In the case of finite fields with arbitrary characteristics, the sponge parameters are expressed in terms of field elements, which leads to the notion of

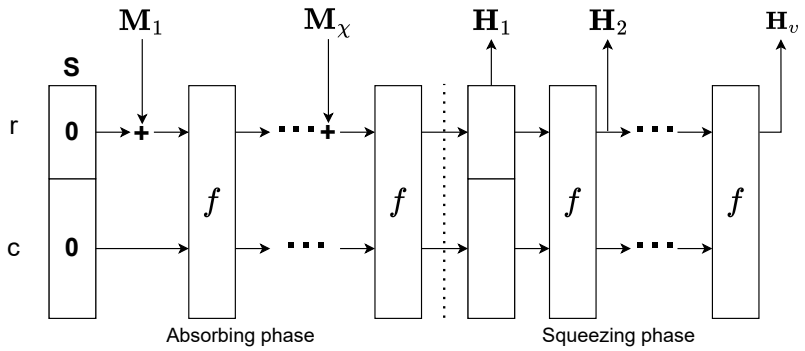


Fig. 1: A sponge function with *rate* r, *capacity* c, and internal permutation f.

arithmetic capacity [26, 27]. In this work, we consider sponges over two characteristics, G and M, and a fixed-length permutation operating on a state size m whose elements lie in the respective finite field.

The sponge function $f : \mathbb{F}_p^m \to \mathbb{F}_p^m$ with rate r (size of the outer part) and capacity c (size of the inner part) where $r + c = m$, takes as input $\mathbf{M}'$ of fixed length, and after generating the padded message $\mathbf{M} \in \mathbb{F}_p^{\chi \cdot r}$, computes the output $\mathbf{H} \in \mathbb{F}_p^{\upsilon \cdot r}$. The sponge function works as follows:

1. Let $\mathbf{S}$ be the state of the sponge function of length $m = r + c$.
2. The state $\mathbf{S}$ is initialized to $(0, \ldots, 0)$.
3. Absorbing phase: Let $\mathbf{M}$ be the padded message with length $\chi \cdot r$ viewed as blocks $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_\chi$ of length r. For each $i \in \{1, 2, \ldots, \chi\}$, $\mathbf{M}_i$ is added to the first r elements of $\mathbf{S}$ and the function f is applied:

$$\mathbf{S} = f(\mathbf{S} + (\mathbf{M}_i || \mathbf{0}))$$

4. Squeezing phase: once all blocks of the padded message have been absorbed, the squeezing phase starts generating the output. In this phase, the function outputs block $\mathbf{H}_1, \ldots, \mathbf{H}_\upsilon$ of length r and updates the internal state $\mathbf{S}$ by applying the function f.

In Figure 1, a schematic construction of the sponge function is illustrated. Assuming that f is computationally indistinguishable from a random permutation, a sponge function with a capacity c offers $(\frac{c \cdot \log_2(\mathrm{p})}{2})$ bits of collision resistance and $(c \cdot \log_2(\mathrm{p}))$ bits of preimage resistance [24, 25].

2.5 Rescue-Prime Optimized

Rescue-Prime Optimised (RPO) is a sponge function instantiated with a permutation over $\mathbb{F}_G$ having 12 state elements. The permutation consists of seven rounds, and each round can be described in terms of four components:

- an S-box $\pi_0 : x \mapsto x^7$;

- an S-box $\pi_1 : x \mapsto x^{\frac{1}{7}}$;
- an MDS matrix M; and
- constant addition $\boxplus\, c$.

With respect to the state vector, $S \in \mathbb{F}_G^{12}$ these components allow to define two types of steps:

- The (F)-step works as follows: first, the S-box π_0 is applied to each state element to provide non-linearity. Then, the state vector is multiplied with an MDS matrix to spread local properties over the entire state. Finally, a different round constant is added to each element to avoid self-symmetry between different rounds;
- The (B)-step is almost the same, only that π_1 is applied instead of π_0. That is, first, the S-box π_1 is applied to each of the state element to provide non-linearity. Then, the state vector is multiplied with an MDS matrix to spread local properties over the entire state. Finally, a different round constant is added to each element to avoid self-symmetry between different rounds.

With this notation, a typical round for a Rescue-like function is (FB). Concretely, RPO can be written as

$$(FB)(FB)(FB)(FB)(FB)(FB)(FB)\,,$$

except that the last M and constant injections are moved to the beginning for reasons explained in [17, Sec. 4.3].

3 New Approaches for Designing AO Primitives

This section introduces new techniques for designing AO ciphers that achieve high security and efficiency. The first is using exponentiation over an extension field. The second idea involves novel security arguments against algebraic attacks, addressing limitations of current methods that bound from below the attack complexity by estimating the time of a Gröbner basis computation.

3.1 S-Boxes Over Extension Field

A core component in AO primitives is the non-linear (S-Box) layer, which is usually a low-degree permutation polynomial over the primitive's underlying field. In most cases, such a polynomial is CCZ-equivalent to a monomial of a small degree (*e.g.,* $x \mapsto x^\alpha, \alpha = 3, 5$). Such representation is crucial for efficiency when the primitive is used in advanced protocols. The non-linear power maps are then composed with linear operations to provide the primitive with a complicated algebraic structure and the required security properties. Reducing the number of such linear and non-linear operations while not jeopardising security is a natural approach for optimising the performance of AO primitives.

Most designs are already aggressively optimised, and no further optimisation is possible without affecting the security margins. Therefore, a new approach is necessary to improve the performance of AO primitives further. In this work, we investigate the possibility of using a non-linear layer applied to a vector of field elements instead of

each field element separately. For example, instead of applying the power map $x_i \mapsto x_i^\alpha$ where x_i is a word in any prime field ($\mathbb{F}_p$), a state element, we define a power map over the cubic extension field ($\mathbb{F}_{p^3}$) of the form

$$X \mapsto X^\alpha. \tag{1}$$

Because there exists an isomorphism between $\mathbb{F}_{p^3}$ and $\mathbb{F}_p^3$, the above map corresponds to:

$$(x_i, x_j, x_k) \in \mathbb{F}_p^3 \mapsto (f_1(x_i, x_j, x_k), f_2(x_i, x_j, x_k), f_3(x_i, x_j, x_k)),$$

where f_1, f_2, f_3 are dense polynomials of degree α. S-Boxes over a vector of words have multiple benefits from the design point of view:

- They have a low degree when represented as polynomials over words, resulting in efficient performance inside the protocols;
- They prevent attackers from controlling single words in a state. Therefore, it is harder to propagate subspaces or other useful patterns;
- The polynomial representation is dense, making the algebraic description of the system more complicated.

3.2 Towards Better Security Arguments Against Algebraic Attacks

The security of AO primitives against algebraic attacks is usually argued using the hardness of solving a multivariate polynomial system modelling the primitive. Using a terminology inspired by [22], we describe the four steps performed to solve a multivariate polynomial system.

SysGen. First, the system of equations is generated. Several heuristics are available to this end, but all of them have to introduce new variables whenever a dth root is used, as it is the compositional inverse of these operations that is of a low degree. This step can further be simplified using techniques from "classical" symmetric cryptanalysis, typically based on the probability one propagation of some affine spaces. This was used to shave off two SPN rounds generically in [28], with the conditions that the S-boxes are monomials over the base field considered,[1] and that the permutation starts with an S-box layer. The system that is ultimately generated is not uniquely defined: different generation strategies will yield different systems. For instance, using affine spaces to simplify the system will remove some equations. Similarly, we can prefer to introduce new variables and equations to obtain more but lower degree equations. The existence of an efficient SysGen procedure is implied by arithmetization-orientation, but it is not necessarily the approach used in attacks.[2]

Note that the algorithms used in the next steps are better understood in the case where the system is expected to have unique (or just a few) solutions. This further adds constraints for the SysGen step, but they are easily handled: the expected number

[1] In contrast to the four proposed permutations — XHash8, XHash12, XHash16 and XHash24 — where monomials are applied over both the base field and the cubic extension field.

[2] In fact, our encoding for XHash12 and XHash24 will be the STARK one, while the one for both XHash8 and XHash16 is significantly different.

of solutions is easily estimated, assuming, *e.g.,* that the hash function behaves like a random function, and the input can be constrained, for instance, by forcing it to be in a vector space of an appropriate dimension.

GröbFind. Once a system is obtained, it is necessary first to endow it with a structure. This, in particular, allows to perform the necessary polynomial arithmetic, *e.g.,* to reduce large degree polynomials modulo lower degree ones. The equations obtained define an ideal: since we investigate their common roots, any linear combination will also have the same roots. An ideal of polynomials has a Gröbner basis. A Gröbner basis is defined for a given *monomial ordering*, that is, a total ordering over the monomials $\prod_i x_i^{\alpha_i}$. Several of these are well-known. In particular, a Gröbner basis in *lexicographic* order can greatly simplify the next steps of the attack. On the other hand, the complexity of finding a Gröbner basis is highly dependent on said ordering. In general, to obtain a basis, *e.g.,* in *grevlex* order, one can use either F4 or F5 [29, 30]. The state-of-the-art approach to compute the complexity in the contexts of AO primitives is the one described in [17], where the behaviour of toy examples is compared to the expected behaviour of a regular system and then extrapolated to larger cases. We performed similar experiments, and the results are reported in Appendix F.

VarElim. Once a Gröbner basis is known, use its structure to extract a univariate polynomial in one of the variables. This extraction step is usually done by reordering the Gröbner basis, *i.e.,* by obtaining a Gröbner basis for a different monomial order, one that is suitable for this purpose. It is usually the *lexicographic* one. This change of order can be done using the FGLM [31] algorithm, whose complexity is well studied, as it boils down to linear polynomial arithmetic. It depends on a quantity called the *degree of the ideal (D_I)* and corresponds to the number of roots the system has in the algebraic closure of the field considered. This number is much higher than the number of solutions in the field itself (typically 1 or 2).

More custom approaches are sometimes possible; for instance, the authors of [22] introduced the combination of MatMul and PolyDet: the idea is to bypass the cost of a full FGLM run by focusing on a single variable that is of particular interest in their case.

UniSolve. By design, we expect a solution to exist. Thus, once a univariate equation is extracted, we solve it using well-known techniques to get the first root of the system. We then substitute its value in the other equations and deduce an assignment for all the variables. This works provided that the equations have the correct structure, but this is always the case for the output of FGLM, under some reasonable assumptions that have been found to hold in practice when attacking AO primitives.

3.2.1 Arguing Security in the Multivariate Case

To show that a hash function is safe from multivariate algebraic attacks, we need to find a lower bound on the time complexity of the resolution of a system corresponding to a Constrained Input Constraind Output (CICO) instance.

Under a reasonable conjecture, we can derive lower bounds on D_I that are sufficient to argue the security of all proposed permutations. To describe this conjecture, we first need the following definition.

Definition 1 (b-Almost Basal.) A system of m polynomials P_i in variables $(x_0, ..., x_{m-1})$ with $b > 0$ is said to be *b-almost basal (b-AB)* if and only if:

- Non-degenerate: the ideal spanned by these polynomials is of dimension 0;
- Basal part: there exists a monomial order such that the leading term of P_i, $i < b$, is the monomial $x_i^{d_i}$ for some integer $d_i > 0$.

Such a system is depicted in Figure 2.

The idea of a B-almost-basal system is to describe a set of multivariate polynomials which is "partially" a Gröbner basis. Indeed, it is sufficient for polynomials to form a Gröbner basis that the leading term (*i.e.,* the highest monomial according to the monomial ordering used) in each P_i is of the form $x_i^{d_i}$. Note in particular that these leading terms involve pairwise distinct variables. In a b-AB system, the first b polynomials have this shape. In fact, an m-AB system is a Gröbner basis since a system with pairwise coprime leading terms in all its equations is known to form a Gröbner basis (see *e.g.,* Proposition 1 of [22]).

$$
b\text{-AB system} \quad \left\{ \begin{array}{l} \left.\begin{array}{l} x_0^{d_0} - P'_0(x_0, \ldots, x_{m-1}) \\ x_1^{d_1} - P'_1(x_0, \ldots, x_{m-1}) \\ \qquad \ldots \\ x_{b-1}^{d_{b-1}} - P'_{b-1}(x_0, \ldots, x_{m-1}) \end{array}\right\} \text{basal part} \\ \\ P_b(x_0, \ldots, x_{m-1}) \\ P_{b+1}(x_0, \ldots, x_{m-1}) \\ \qquad \ldots \\ P_{m-1}(x_0, \ldots, x_{m-1}) \end{array} \right.
$$

Fig. 2: The structure of an b-AB system, where $P_i(x) = x_i^{d_i} - P'(x)$.

Using this notion, we are ready to state the following conjecture.

Conjecture 1 (Monotonous Ideal Degree Conjecture (MIDC).) For a b-AB system, the degree D_I of the ideal is lower bounded by $D_I \geq \prod_{i=0}^{b-1} d_i$.

The intuition behind this conjecture is simple: the ideal degree corresponds to the number of solutions the system has in the algebraic closure of the underlying field, and this conjecture postulates that this number does not decrease when we take into account the last equations. That is where the term "monotonous" comes from: we assume that the ideal degree increases or stays constant but does not decrease when

we consider more equations within the system. To put it differently, this conjecture posits that the actual Gröbner basis of the system contains the basal part and that the last equations (those not in the basal part) do not weaken the algorithm.

This conjecture is true when $b = m$: in this case, the system is a Gröbner basis, and the ideal degree is known to be $\prod_{i=0}^{m-1} d_i$.

4 Design Rationale Of The XHash Family

So far, three main components have been used in Marvellous designs: S-boxes, linear layers, and injections of round constants. Together, these result in a dense polynomial representation of a high degree. Building on Rescue's S-boxes, the designers of RPO fixed the MDS matrix to one that can be efficiently computed over the Goldilocks prime field $\mathbb{F}_G$ *i.e.,* $G = 2^{64} - 2^{32} + 1$. A natural question then arises — whether the efficiency can be improved any further without sacrificing security. Noting that the costs of π_0 and constant injection are almost negligible and that M has been aggressively optimised, the remaining bottleneck has been observed to be the π_1 S-box. On a CPU, each call to π_1 requires about 70 multiplications; there are 12 calls in each (B)-step, and seven (B) steps, totalling in $70 \cdot 12 \cdot 7 = 5,880$ multiplications. This is by far the largest cost driver in the entire permutation.

Before attempting to reduce this cost, one must determine what purpose the (B)-step serves. Citing [17], the two S-boxes (π_0, π_1) are motivated as follows:

The difference between π_0 and π_1 is in their degree. They should be chosen such that π_1 has a high degree when evaluated forward (i.e., in the direction of the encryption) and a low degree when evaluated backwards (i.e., in the direction of the decryption). The other S-box, namely π_0, is chosen with the opposite goal (i.e., to have a low degree in the forward direction and a high degree in the backward direction). This choice serves to achieve three objectives: (i) no matter which direction an adversary is trying to attack, the degree is guaranteed to be high; (ii) it results in the same cost for the encryption and decryption functions, and (iii) owing to non-procedural computation, the low-degree representation of each S-box can be evaluated efficiently.

We remind the reader that the stated goal of [17] was to design a general-purpose primitive usable not only for hashing or not only in STARKs but also to work with any sufficiently large prime field. Goal (ii) is irrelevant since we expect honest users only to compute the primitive "forward", never requiring efficient "backward" evaluation. Hence, we can abandon Goal (ii) altogether.

For similar reasons, goal (i) can also be relaxed. Avoiding attacks still requires that the degree in either direction be high — but it no longer needs to be the same in both directions. First, observe that the high degree in the "backward" direction is ensured by the (F)-rounds, which we are not looking to optimise. Therefore, the degree remains sufficiently high, and the security argument does not need to be re-evaluated.

We now conjecture the possibility that the (B)-step may offer "too much security" for our purposes. Concretely, [17] observes that the (B)-step achieves maximal degree already after two rounds. Given that the minimal number of rounds is set to ten in Rescue, eight in Rescue-Prime, and seven in RPO, positing a (B')-step that can

achieve, hypothetically speaking, a maximal degree in three rounds, the overall performance can be improved without affecting security. Getting ahead of ourselves, this is exactly what we will be doing. To reduce the cost of achieving a compact polynomial description, we propose to use power maps over an extension field. Exponentiation in an extension field simultaneously provides diffusion by mixing the base field elements as well as confusion by doing so in a non-linear way.

Revisiting the design rationale for general Marvellous design, we see that in the context of a hash function operating over state elements in a prime field, the interpolation attack would be the main concern in case the polynomial degree is not sufficiently high. Considering in detail the argument against the interpolation attack, we see that resistance is achieved when the univariate polynomial describing the cipher is:

1. dense; and
2. of maximal degree.

Intuitively, the composition of an MDS matrix (which ensures optimal diffusion) with a power map ensures density due to the multinomial theorem.

We are left to ensure a maximal degree. Considering that

$$x^{\frac{1}{\alpha}} = x^{\frac{(2G-1)}{\alpha}} \in \mathbb{F}_G$$

achieves an almost maximal degree or close to that already in a single call to π_1, we can informally argue that this is an "overkill" and that an S-box resulting in a lower degree per step may still be sufficient to resist the interpolation attack as long as it reaches a maximal degree or close to that within a reasonable number of rounds. Thus, we are looking for a power map β such that $\alpha < \beta < \frac{2G-1}{\alpha}$. However, by definition, such an S-box will be inefficient to compute inside the STARK, both directly and folded.

Seeing that better power maps are not readily available, we can instead apply the same power maps, but to fewer elements. However, with this approach, only the elements to which the power map was applied are "protected". If this approach is taken, the partial S-box layer must be complemented with another operation that would mix high-degree elements with the rest non-linearly. If done right, this would ensure that all polynomials are both dense and of high degree.

4.1 Specification of XHash

We complement the S-boxes (π_0, π_1) described in Section 2.5 with a third type of S-box:

- π_2 is similar to π_0 in that it takes a field element and raises it to the αth power. However, this time, the element is in the cubic extension field rather than the base field.

Using the new S-box π_2, we define four permutations: XHash8, XHash12, XHash16, and XHash24. The polynomial representation of π_2 for the Goldilocks prime field $\mathbb{F}_G$ and for the Mersenne prime field (M31) $\mathbb{F}_M$ is given in Appendix D and Appendix E, respectively.

XHash12 and XHash24 use a full layer of π_1 S-boxes applied to all elements of the state, while XHash8 and XHash16 further improving efficiency by applying a partial layer of π_1 S-boxes to two out of every three elements of the state, where the total number of state elements is a multiple of three.

4.1.1 XHash12

XHash12 employs a full layer of π_1 S-boxes and works as follows:

- A version of the (F)-step starting with round constants injection, followed by an MDS multiplication, and concluded by applying π_0 to each word of the state;
- A version of the (B)-step starting with MDS multiplication, followed by round constants injection, and concluded by applying π_1 to each word of the state;
- A $(P3)$-step starting with constant injection, then π_2 given in Equation 1, is applied in the cubic extension field, and after π_2 the state is decomposed back into the base field;
- The last step is a special one, denoted by (MC). It consists of an MDS multiplication followed by round constant injection.

Based on the XHash12 permutation, we propose a new hash function, efficient for ZK-STARKs over the Goldilocks prime field named **XHash12-Goldilocks**. It has 12 state elements, and $\alpha = 7$, which is the smallest possible permutation power map over the Goldilocks field. This 12-element state in the $(P3)$-step is restructured as a 4-element vector in a cubic extension field, *i.e.,*

$$(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}) \mapsto (S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11}),$$

where each $s_{i,j,k} \in \mathbb{F}_{G^3}$. This is followed by an application of π_2 to each of these extension field elements, such that

$$(S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11}) \mapsto (S^7_{0,1,2}, S^7_{3,4,5}, S^7_{6,7,8}, S^7_{9,10,11}).$$

At this point, the state is decomposed back into a 12-element vector in $\mathbb{F}_G$; as depicted in Figure 3a.

4.1.2 XHash24

XHash24 employs a full layer of π_1 S-boxes, and works as follows:

- A version of the (F)-step starting with round constants injection, followed by an MDS multiplication, and concluded by applying π_0 to each word of the state;
- A version of the (B)-step starting with MDS multiplication, followed by round constants injection, and concluded by applying π_1 to each word of the state;
- A $(P3)$-step starting with constant injection; then, π_2 is applied in the cubic extension field, and after π_2 the state is decomposed back into the base field;
- The last step is a special one, denoted by (MC). It consists of an MDS multiplication followed by round constant injection.

Based on the XHash24 permutation, we propose a new hash function, efficient for Circle-STARKs over the Mersenne prime field named **XHash24-M31**. It consists of 24 state elements with $\alpha = 5$, which is the smallest possible permutation power map over M31. This 24-element state in the $(P3)$-step is restructured as an 8-element vector in a cubic extension field, *i.e.,*

$$(s_0, s_1, s_2, \ldots, s_{21}, s_{22}, s_{23}) \mapsto (S_{0,1,2}, \ldots, S_{21,22,23}),$$

where each $s_{i,j,k} \in \mathbb{F}_{M^3}$. This is followed by an application of π_2 to each of these extension field elements, such that

$$(S_{0,1,2}, \ldots, S_{21,22,23}) \mapsto (S^5_{0,1,2}, \ldots, S^5_{21,22,23}).$$

At this point, the state is decomposed back into a 24-element vector in $\mathbb{F}_M$; as depicted in Figure 4a.

4.1.3 XHash8: With Partial $x^{\frac{1}{7}}$ Layer

XHash8 is a more aggressively optimised version of XHash12. The difference is that it employs partial layers of π_1 S-boxes and works as follows:

- A version of the (F)-step starting with a constant addition, followed by an MDS multiplication, and concluded by applying π_0 to each word of the state;
- A modified version of the (B)-step, namely the $(B^{'})$-step, starting with MDS multiplication, followed by constant addition, and concluded by applying a partial layer of π_1 S-boxes to two out of every three elements of the state, where the total number of state elements is 12;
- A $(P3)$-step starting with constant injection; then, π_2 is applied in the cubic extension field, and after π_2 the state is decomposed back into the base field;
- The last step is a special one, denoted by (MC). It consists of an MDS multiplication followed by round constant injection.

Based on the XHash8 permutation, we propose a new hash function, efficient for ZK-STARKs over the Goldilocks prime field named **XHash8-Goldilocks**. In the $(B^{'})$-step, π_1 is applied to 8 out of the 12 state elements:

$$(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}) \mapsto (s_0^{\frac{1}{7}}, s_1, s_2^{\frac{1}{7}}, s_3^{\frac{1}{7}}, s_4, s_5^{\frac{1}{7}}, s_6^{\frac{1}{7}}, s_7, s_8^{\frac{1}{7}}, s_9^{\frac{1}{7}}, s_{10}, s_{11}^{\frac{1}{7}}),$$

and this 12-element state in the $(P3)$-step is restructured as a 4-element vector in a cubic extension field, *i.e.,*

$$(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}) \mapsto (S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11}),$$

where each $s_{i,j,k} \in \mathbb{F}_{G^3}$. This is followed by an application of π_2 to each of these extension field elements, such that

$$(S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11}) \mapsto (S^7_{0,1,2}, S^7_{3,4,5}, S^7_{6,7,8}, S^7_{9,10,11}).$$

At this point, the state is decomposed back into a 12-element vector in F_G as depicted in Figure 3b.

4.1.4 XHash16: With Partial $x^{\frac{1}{5}}$ Layer

XHash16 is a more aggressively optimised version of XHash24. The difference is that it employs partial layers of the π_1 S-boxes and works as follows:

- A version of the (F)-step starting with a constant addition, followed by an MDS multiplication, and concluded by applying π_0 to each word of the state;
- A modified version of the (B)-step, namely the $(B^{'})$-step, starting with MDS multiplication, followed by constant addition, and concluded by applying a partial layer of π_1 S-boxes to two out of every three elements of the state, where the total number of state elements is 24;
- A $(P3)$-step starting with constant injection; then, π_2 is applied in the cubic extension field, and after π_2 the state is decomposed back into the base field;
- The last step is a special one, denoted by (MC). It consists of an MDS multiplication followed by round constant injection.

Based on the XHash16 permutation, we propose a new hash function, efficient for the Circle-STARKs over the M31 named **XHash16-M31**. In the $(B^{'})$-step, π_1 is applied to 16 out of the 24 state elements:

$$(s_0, s_1, s_2, \ldots, s_{21}, s_{22}, s_{23}) \mapsto (s_0^{\frac{1}{5}}, s_1, s_2^{\frac{1}{5}}, s_3^{\frac{1}{5}}, s_4, s_5^{\frac{1}{5}}, \ldots, s_{18}^{\frac{1}{5}}, s_{19}, s_{20}^{\frac{1}{5}}, s_{21}^{\frac{1}{5}}, s_{12}, s_{23}^{\frac{1}{5}}),$$

and this 24-element state in the $(P3)$-step is restructured as an 8-element vector in a cubic extension field, *i.e.,*

$$(s_0, s_1, s_2, \ldots, s_{21}, s_{22}, s_{23}) \mapsto (S_{0,1,2}, \ldots, S_{21,22,23}),$$

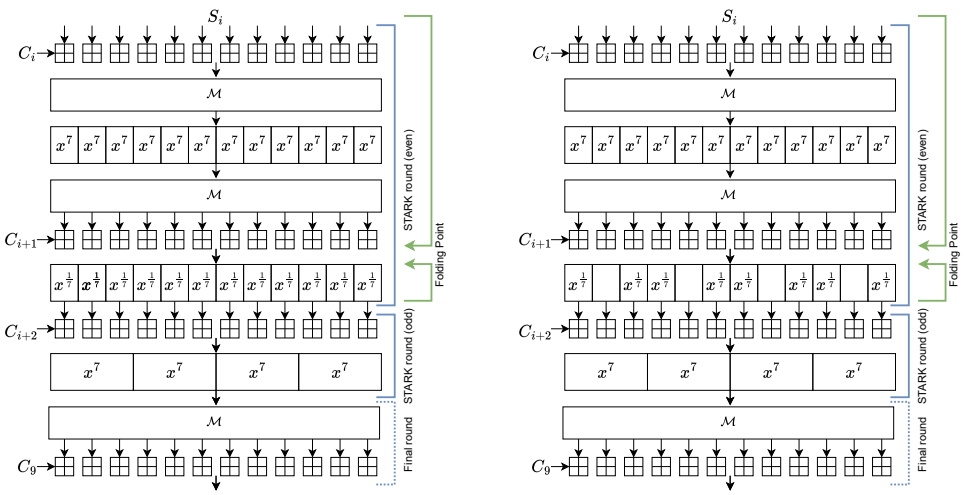
where each $s_{i,j,k} \in \mathbb{F}_{M^3}$. This is followed by an application of π_2 to each of these extension field elements, such that

$$(S_{0,1,2}, \ldots, S_{21,22,23}) \mapsto (S_{0,1,2}^5, \ldots, S_{21,22,23}^5).$$

At this point, the state is decomposed back into a 24-element vector in F_M as depicted in Figure 4b.

4.2 Claimed Security

XHash12-Goldilocks and XHash8-Goldilocks offer $(4 \cdot \log_2(2^{64} - 2^{32} + 1))/2 \approx$ **128**-bit security, obtained by using either permutation in a sponge construction with the elements $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$ as the outer part and $(s_8, s_9, s_{10}, s_{11})$ as the inner part. Similarly, XHash24-M31 and XHash16-M31 offer $(8 \cdot \log_2(2^{31} - 1))/2 \approx$ **124**-bit security, obtained by using either permutation in a sponge construction with the elements $(s_0, s_1, \ldots, s_{14}, s_{15})$ as the outer part and $(s_{16}, s_{17}, \ldots, s_{22}, s_{23})$ as the inner part. Both formulas follows from the sponge's generic security arguments. For more detail, we refer the readers to [25].



(a) XHash12-Goldilocks

(b) XHash8-Goldilocks

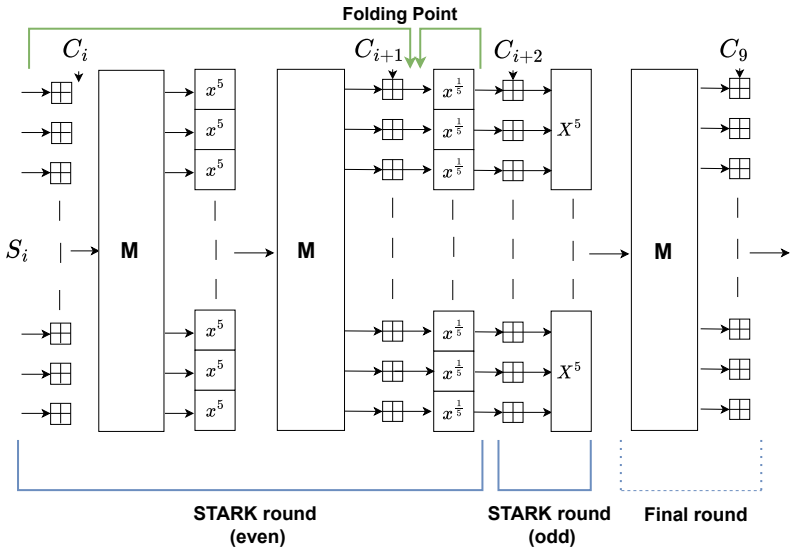
Fig. 3: The XHash12 and XHash8 permutations. The folding point depicts how to compress the (F) and (B) rounds into a single low-degree polynomial. This allows the verifier to evaluate a sequence of $(FB)(P3)$ steps as two STARK rounds (blue lines). The dotted blue line represents the finalisation step.

4.3 Round Constants

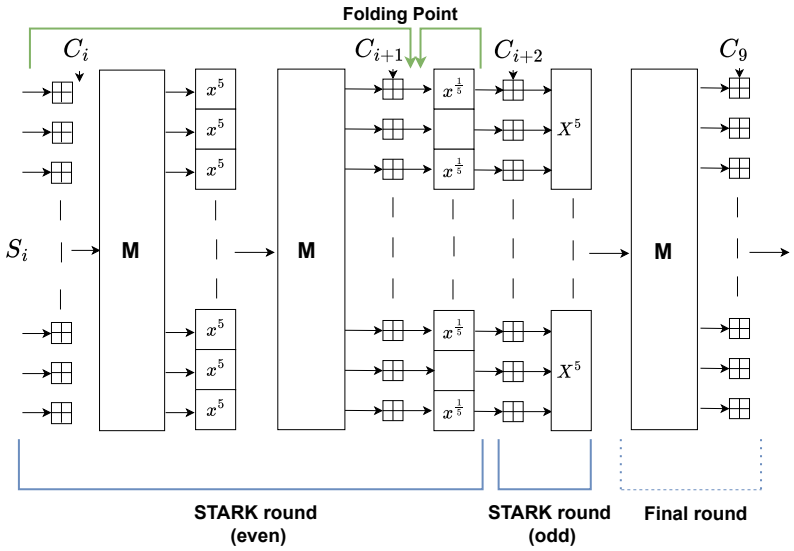
To ensure "nothing-up-my-sleeve" round constants, we reuse the ones derived for RPO. Recall that the RPO constants were derived by employing the following procedure:

- Start from the string `RPO(%i,%i,%i,%i)`;
- Populate the wildcards "`%i`" with the ASCII decimal expansion of the integer parameters (prime, state elements (m), *capacity* (c), bit security (s)), in that order;
- Use SHAKE256 to expand this ASCII string into $(9 \cdot 3 \cdot 6 \cdot m)$ pseudorandom bytes;
- For every chunk of 9 bytes, compute the matching integer by interpreting the byte array as the integer's base-256 expansion with the least significant digit first;
- Reduce the obtained integer modulo the prime;
- Collect all such integers. The list of obtained field elements constitutes the list of round constants.

This method ensures that constants are generated transparently and reliably, thereby avoiding arbitrary or hidden selections. The code for generating the constants can be found in [32] and the round constants are given in Appendix B.



(a) XHash24-M31



(b) XHash16-M31

Fig. 4: The XHash24 and XHash16 permutations. The folding point depicts how to compress the (F) and (B) rounds into a single low-degree polynomial. This allows the verifier to evaluate a sequence of $(FB)(P3)$ steps as two STARK rounds (blue lines). The dotted blue line represents the finalisation step.

4.4 MDS Matrix

XHash12-Goldilocks and XHash8-Goldilocks, use the same MDS matrix found by the RPO project. That is, a circulant matrix whose first row is

$$[7, 23, 8, 26, 13, 10, 9, 7, 6, 22, 21, 8].$$

For XHash24-M31 and XHash16-M31, we generate an efficient MDS in the Circle-STARKs setting by employing the method suggested in [33, 34]. We start by constructing a circulant MDS matrix with 32 elements. Then, we unroll this matrix to its full 32×32 form and truncate the last eight columns and the last eight rows. The complete procedure can be found in Appendix C, where we also provide a first row of the circulant matrix.

4.5 Number of Rounds (N)

In [17, page 18], the authors explain how a safe number of rounds for a Marvellous design should be selected. According to this, the number of rounds should follow $2\lceil \max(l_0, l_1, 5) \rceil$, where l_0 is the maximal number of rounds that can be attacked by any statistical and algebraic attack except the Gröbner basis (GB) attack, l_1 is the maximal number of rounds that can be attacked by a GB attack, and 5 is a sanity factor. The safety margin was later relaxed in [19] to be $(1.5)\lceil \max(l_0, l_1, 5) \rceil$, and, seeing that no attacks were suggested over a couple of years, the number for RPO [20] was set to 7 *i.e.,* $(1.5)\lfloor \max(l_0, l_1, 5) \rfloor$.

Table 1 in [17, page 20] shows different formulas to calculate the safe number of rounds against differential cryptanalysis (DC) and the interpolation attack. In addition, for the GB attack, a formula is given in [17, page 30]. Consequently, we decided on 6 rounds for the XHash family with one linear layer at the end. Therefore, each permutation consists of 7 steps in total, described as follows:

$$(FB)(P3)(FB)(P3)(FB)(P3)(MC),$$

or

$$(FB^{'})(P3)(FB^{'})(P3)(FB^{'})(P3)(MC).$$

4.6 Padding

For hashing field elements, we use the zero-padding scheme suggested in [25]. Specifically, if the length of the last block is smaller than $r = 8$ field elements in both XHash12-Goldilocks and XHash8-Goldilocks, or $r = 16$ field elements in both XHash24-M31 and XHash16-M31, a sufficient number of [0] elements are appended to complete it.

We note that this padding rule alone is not sponge compliant. To ensure bijectivity, we partition the input space into eight input domains for the Goldilocks prime field and 16 input domains for the Mersenne prime field. All messages whose last block is of length 8 in $\mathbb{F}_G$ and 16 in $\mathbb{F}_M$, are assigned to the 0-domain; all messages whose last block is of length 7 in $\mathbb{F}_G$ and 15 in $\mathbb{F}_M$, are assigned to the 1-domain, and so on.

To enforce this domain separation, the 9th state element in $\mathbb{F}_G$ and the 17th state element in $\mathbb{F}_M$, that is, the first element of the inner part are initialised to the domain identifier.

5 Security Rationale Of The XHash Family

The security of all proposed hash functions XHash8-Goldilocks, XHash12-Goldilocks, XHash16-M31, and XHash24-M31 is analysed against standard attacks, including statistical and algebraic attacks.

5.1 Statistical Attacks

Statistical attacks (*e.g.,* DC) have been a source of trouble in the design of traditional symmetric algorithms (*i.e.,* non-AO algorithms). Early work in the area of AO primitives observed that in this setting even a small number of rounds is enough to thwart them. Nevertheless, for completeness, we provide arguments based on the wide-trail strategy.

5.1.1 Differential cryptanalysis (DC)

We analyse the resistance of XHash8-Goldilocks and XHash16-M31 against differential cryptanalysis by counting the minimum number of active S-boxes. The resistance of both XHash8-Goldilocks and XHash16-M31 also implies the resistance of XHash12-Goldilocks and XHash24-M31. Since, for any trail pattern, XHash12-Goldilocks and XHash24-M31 activate more S-boxes than XHash8-Goldilocks and XHash16-M31. Since the S-box structure is the same within the Goldilocks-based designs (XHash12 and XHash8) and likewise within the Mersenne-based designs (XHash24 and XHash16), the probability of the best differential characteristic for XHash8-Goldilocks and XHash16-M31 bounds from above for XHash12-Goldilocks and XHash24-M31, respectively.

The analysis follows the standard argument: we find a lower bound on the number of active S-boxes and an upper bound on the probability of the best differential transition in any S-box; the quantity obtained from raising the latter to the power of the former upper bounds the probability of the best differential characteristic.

The aspects of our new functions that require special care are:

1. The $(B^{'})$-step uses a partial S-box layer;
2. The $(P3)$-step mixes base field elements in a non-linear way.

As a result of these particularities, the Two-Round Propagation Theorem originally stated by Daemen and Rijmen in [23, Theorem 9.3.1] for the case of alternating block ciphers cannot be applied directly to both XHash8-Goldilocks and XHash16-M31.

To address these observations, we provide a detailed step-by-step analysis about the minimum number of active S-boxes for XHash8-Goldilocks in Table 1 and for XHash16-M31 in Table 2. First, we observe that in the first step, the adversary controls only the outer part of the sponge and can therefore introduce a difference in $1 \leq d_{(I)} \leq r = 9$ field elements for XHash8-Goldilocks, and $1 \leq d_{(I)} \leq r = 17$ field elements for

XHash16-M31. Consequently, after the application of the MDS matrix, $4 \leq d_{(F)} \leq 12$ S-boxes are active in the (F)-step for XHash8-Goldilocks and $8 \leq d_{(F)} \leq 24$ S-boxes are active in the (F)-step for XHash16-M31.

For the $(B^{'})$-step, the adversary can activate $0 \leq d_{(B')} \leq 5$ S-boxes for XHash8-Goldilocks and $0 \leq d_{(B')} \leq 9$ S-boxes for XHash16-M31, followed by $1 \leq d_{(P3)} \leq 3$ S-boxes in the $(P3)$-step for XHash8-Goldilocks and $1 \leq d_{(P3)} \leq 6$ S-boxes in the $(P3)$-step for XHash16-M31.[3]

In total, over a triplet of consecutive $(FB^{'})(P3)$ steps in XHash8-Goldilocks, at least nine S-boxes are activated from (π_0, π_1), and at least one S-box of type π_2. In the case of XHash16-M31, at least 17 S-boxes are activated from (π_0, π_1), and at least one S-box of type π_2.

In the case of XHash8-Goldilocks, the upper bound probability of a differential characteristic is estimated to be 2^{-61} for the $(FB^{'})$ and 2^{-186} for the $(P3)$. Therefore, for the 128-bit security, the probability of an optimal differential transition across a triplet $(FB^{'})(P3)$ is upper bounded by $2^{9\cdot(-61)} \cdot 2^{1\cdot(-186)} = 2^{-549-186} = 2^{-735}$.

In the case of XHash16-M31, the upper bound probability of a differential characteristic is estimated to be 2^{-29} for the $(FB^{'})$ and 2^{-89} for the $(P3)$. Therefore, for the 124-bit security, the probability of an optimal differential transition across a triplet $(FB^{'})(P3)$ is upper bounded by $2^{17\cdot(-29)} \cdot 2^{1\cdot(-89)} = 2^{-493-89} = 2^{-582}$.

Theorem 1 *Let $\mathbb{F}_q$ be a finite field of order $q = p^n$ and characteristic p. Let $F(x) = x^\gamma$ be a power map defined over $\mathbb{F}_q$, then F is differentially $(\gamma - 1)$-uniform.*

Proof Given $\alpha, \beta \in \mathbb{F}_q, \alpha \neq 0$, the cardinality of the set $\mathcal{D} = \{x \in \mathbb{F}_q |\ F(x + \alpha) - F(x) = \beta\}$ can be computed as the number of roots for the following polynomial:

$$(x + \alpha)^\gamma - x^\gamma = \sum_{i=0}^{\gamma-1} \binom{\gamma}{i} \alpha^{\gamma-i} x^i, \tag{2}$$

which is a polynomial of degree $\gamma - 1$. It is well known that a polynomial of degree α has at most α roots. We, therefore, obtain the upper bound $|\mathcal{D}| \leq \gamma - 1$. □

[3] We note that the S-boxes in the $(P3)$-step are of different type than those in the (FB)-step.

Input	M_F	π_0	$M_{B'}$	π_1	π_2
1	12	12	1	0	1
2	11	11	2	0	1
3	10	10	3	0	1
4	9	9	4	0	2
5	8	8	5	1	2
6	7	7	6	2	2
7	6	6	7	3	3
8	5	5	8	4	3
9	4	4	9	5	3

Table 1: Minimum number of active S-boxes at each stage in a round of XHash8-Goldilocks

Input	M_F	π_0	$M_{B'}$	π_1	π_2
1	24	24	1	0	1
2	23	23	2	0	1
3	22	22	3	0	1
4	21	21	4	0	2
5	20	20	5	0	2
6	19	19	6	0	2
7	18	18	7	0	3
8	17	17	8	0	3
9	16	16	9	1	3
10	15	15	10	2	4
11	14	14	11	3	4
12	13	13	12	4	4
13	12	12	13	5	5
14	11	11	14	6	5
15	10	10	15	7	5
16	9	9	16	8	6
17	8	8	17	9	6

Table 2: Minimum number of active S-boxes at each stage in a round of XHash16-M31

5.2 Algebraic Attacks

Arguing security against algebraic attacks is a complicated task, as these are not so well understood at this stage, and the interplay between the "algebraic" part and more classical techniques can sometimes be used to an attacker's advantage in lowering the overall complexity of the attack.

By "algebraic" attack, we refer to attacks that ultimately culminate with the resolution of a (system of) equation(s), of which the roots need to be found. These can rely on a single univariate equation or multiple multivariate equations, as discussed in Section 3.2.

5.2.1 Univariate Algebraic Attacks

In this section, we argue the resistance of our new designs against algebraic attacks. We investigate two properties of the polynomial modelling: (1) polynomial degree and (2) density. These two properties are known to be linked to the efficiency of algebraic attacks.

Polynomial Degree.

We analyse the polynomial degree of XHash8-Goldilocks and XHash16-M31, and use them as a lower bound for the polynomial degree of XHash12-Goldilocks and XHash24-M31.

Similar to other algorithms from the Marvellous family, a high polynomial degree is achieved by applying a large power map to the elements of the state. However, when using XHash8-Goldilocks or XHash16-M31, π_1 is applied only to part of the state. Even if we ignore the $(P3)$ round, the subsequent application of M is expected to distribute the high-degree terms across the entire state. However, this is difficult to formalise without resorting to complex case analysis, and furthermore, it is not evident that a linear transformation alone can sufficiently propagate algebraic properties.[4] Thus, we do not abstract the $(P3)$-step but instead analyse its diffusion properties directly.

Let $(x_0, x_1, x_2) \in F_G^3$ and $x_{0,1,2} \in F_{G^3}$. For brevity, we consider only a single squaring operation. Consider the three polynomials describing the base field elements after a single squaring operation:

$$\begin{aligned} x_{0,1,2}^2 = (&c_0 x_0^2 + c_1 x_1 \cdot x_2, \\ &x_1 \cdot (c_3 x_0 + c_4 x_2) + c_5 x_2^2, \\ &c_6 x_0 \cdot x_2 + c_7 x_1^2 + c_8 x_2^2), \end{aligned}$$

where $c_i \in \mathbb{F}_G$ for all $0 \leq i \leq 8$. Taking into account that $x_0 = \pi_1(y_0) = y_0^{1/7}, x_2 = \pi_1(y_2) = y_2^{1/7}$ and setting, $x_1 = y_1$ we get

$$\begin{aligned} x_{0,1,2}^2 = (&y_0^{2/7} + y_1 \cdot y_2^{1/7}, \\ &y_1 \cdot (y_0^{1/7} + y_2^{1/7}) + y_2^{2/7}, \\ &y_0^{1/7} \cdot y_2^{1/7} + y_1^2 + y_2^{2/7}), \end{aligned}$$

and conclude that every possible polynomial description of the initial state is of a high degree even before applying the MDS matrix. While this observation is already sufficient to argue resistance against interpolation attacks, in Appendix D we present the complete case for x^7 in F_{G^3} and in Appendix E for x^5 in F_{M^3}.

Density.

Several works have recently noticed that the solving degree of a polynomial system describing an AO primitive and consisting of π_0 S-boxes alone is smaller than the degree of regularity. In particular, Sauer observed in [37] that for Rescue-like functions (*i.e.,* functions mixing π_0 and π_1 S-boxes), the solving degree grows at the same rate as the Macaulay bound, whereas this is not the case for Hades-like ciphers. This observation was later independently confirmed by Ashur, Kindi, Meier, Szepieniec, and Threadbare in the design of RPO and more recently quantified and leveraged into an attack by Ashur, Buschman, and Mahzoun in [36].

To explain this observation, Sauer introduces a new metric, "involvement," as a proxy for the difficulty of finding a Gröbner basis [37]. Two notions of involvement are suggested based on vectors of origin: one based on the normalised average of polynomials from the original system required to describe each element in the Gröbner basis and the other on the number of non-zero coefficients in the matrix describing the

[4] See [35] for the binary case and [36] for the prime case.

vectors of origin. Sauer presents heat maps suggesting that the latter is a reasonable, even if noisy, proxy for the difficulty of finding a Gröbner basis.

Here, we suggest that the notion of density is sufficient to capture the quality we are interested in when designing a symmetric-key primitive. As an instructive example, consider the polynomial modeling of a single Rescue round consisting of consecutive (FB) steps:

$$\begin{aligned}&\left(\sum_{k=0}^{m-1} M[j,k]S_{2i-1}[k]^{\alpha}\right) - \\ &\left(\sum_{k=0}^{m-1} M^{-1}[j,k](S_{2i+1}[k] - K_{2i+1}[k])\right)^{\alpha} + K_{2i}[j] = 0.\end{aligned} \tag{3}$$

Note that in the second term, the power map α is applied to the entire sum, creating a complicated expression due to the multinomial theorem. For $\alpha = 7$ and $m = 12$ as in the case of RPO, each such polynomial consists of $12 + \binom{18}{11}$ monomials in 24 variables. Comparably, the modeling of an (F)round of RPO is

$$\left(\sum_{k=0}^{m-1} M[j,k]S_{i-1}[k]^{\alpha}\right) + K_{2i}[j] - S_i[j] = 0, \tag{4}$$

which consists of $12+1 = 13$ monomials in $12+1 = 13$ variables. It is straightforward to see that polynomials in Equation 3 are denser than polynomials in Equation 4 and since clearly density implies involvement (but not necessarily the other way around), we conjecture density to be the explanation for Sauer's.[5]

XHash8-Goldilocks achieves density differently. Consider an even STARK round consisting of one pair of $(FB^{'})$ steps. This round gives rise to two types of polynomials:

$$\left(\sum_{k=0}^{m-1} M[j,k]\left(\sum_{k=0}^{m-1} M[j,k]\left[S_{i-1}[j] + K_i[j]\right]\right)^{\alpha}\right) + K_i^{'}[j] - S_i^{\alpha}[j] = 0, \quad j \not\equiv 1 \pmod 3$$
$$\left(\sum_{k=0}^{m-1} M[j,k]\left(\sum_{k=0}^{m-1} M[j,k]\left[S_{i-1}[j] + K_i[j]\right]\right)^{\alpha}\right) + K_i^{'}[j] - S_i[j] = 0, \quad j \equiv 1 \pmod 3$$

neither of which is dense. Density arises from the polynomial modeling of $(P3)$. The polynomial description of $(P3)$ can be found in Appendix D showing that each base element can be modeled as a 3-variate polynomial of degree seven with 31–34 monomials. Following the MDS, each state element is modeled by a polynomial with $36 \cdot 4 = 144$ monomials in 13 variables.

Similarly, the polynomial description of $(P3)$ in the case of XHash16-M31 is given in Appendix E, shows that each element has degree five with 7–8 monomials. After the

[5] This conjecture arises independently in the argument of FreeLunch [22] attacks.

application of MDS, each state element is modeled by a polynomial with $7 \cdot 3 \cdot 8 = 168$ monomials in 25 variables.

5.2.2 Multivariate Algebraic Attacks

Our security arguments are based on the technique and conjecture given in Section 3.2.1. We only present a summary here, and the details are expanded upon in the appendices. More precisely, the modeling strategy is detailed in Appendix G.1, the custom monomial ordering we rely on can be found in Appendix G.2, and our experimental results are described in Appendix G.3.

Our attack is again intended to find solutions for the CICO problem. To this end, we generate equations by introducing c variables x_i corresponding to the inputs of the permutation, c is the number needed to obtain a system expected to have a unique solution (we aim to fix c outputs to 0 in the end). We then let the other inputs of the permutation be 0 and let the multivariate polynomials describing the content of the state elements evolve as the rounds are iterated, adding new equations to the system each time a π_0 S-Box is encountered in both XHash8 and XHash12 permutations or a 5th root is encountered in both XHash16 and XHash24 permutations. In that case, we introduce a new variable y_i, and add to our system the equation stating that y_i^7 must be equal to the input of the 7th root and y_i^5 must be equal to the input of the 5th root.

For XHash12-Goldilocks and XHash24-M31, we obtain only equations corresponding to new variables y_i, but not for XHash8-Goldilocks and XHash16-M31. In the latter case, we get the *FreeLunch* equations described in [22]. We then naturally borrow the monomial ordering. We describe it in detail in Appendix G.2, but in a nutshell, the idea is to assign a weight multiplied by α^{2i} in each round of the permutation to the variables introduced at round i. This ensures that the new variables are the leading terms in the equation where they are introduced. Finally, we complete the system by adding final equations corresponding to the constraints that c output words must be set to 0.

A system built in this way has a basal part corresponding to all the equations in which new variables are introduced, *i.e.,* it has a basal part consisting of as many equations as there are α^{th} roots in the round permutation, each equation being of degree α. In other words, all equations but the last ones (those encoding the output constraint) are in the basal part. As a consequence, the MIDC implies that the ideal degree of the full system should be at least α raised to a power corresponding to the number of αth roots.

We were able to implement the generation of such systems against round-reduced instances of the XHash family. This allows us to see that, indeed, the MIDC conjecture holds in practice for these systems: we do have that D_I is lower-bounded by the product of the degrees of the equations in the basal part. In fact, in our experiments, we found that $D_I = \prod_{i=0}^{m-1} d_i$, even though the last polynomials have identical leading terms. In that case, the last equations participate as if they had different leading terms. Still, we deem it safer at this stage to only take the basal part into account when making a security claim.

Applying our attack strategy against one word in XHash8-Goldilocks, *i.e.,* solving a CICO instance with one [0] in the input and one in the output, we simply obtain a

FreeLunch [22] system. It is a particular case of an m-AB system. Under the MIDC, the ideal degree corresponding to an attack on w words is lower bounded by the complexity of attacking a single word. Indeed, when $w > 1$, the first equations still form a basal part of the polynomial system, only the last $w - 1$ equations are not basal. As a consequence, we get a bound on D_I which corresponds to its value in the $w = 1$ case, namely $D_I \geq d^{8r}$.

For XHash8-Goldilocks, even ignoring the presence of the first equation with the leading term x_0 among the final equations, we get that $D_I \geq d^{3\times 8} \approx 2^{67}$. Adding the equation in x_0, which also satisfies the criteria to be part of the basal part, we get $D_I \geq d^{3\times 8} \times d^{3\times 2} \approx 2^{84}$. As a consequence, for GB-based algebraic attacks to be a threat against XHash8-Goldilocks, we would need two things:

1. that the bound for D_I is tight (experiments indicate that it is not), and
2. an FGLM-like algorithm with a complexity strictly sub-quadratic in D_I — in fact, even a complexity proportional to $D_I^{1.5}$ would give a complexity around 2^{126} in this case: barely an attack.

For XHash12-Goldilocks, we get a bound of $D_I \geq d^{3\times 12} \approx 2^{101}$, which is even higher and thus safer. For XHash16-M31, we get a bound of $D_I \geq d^{3\times 16} \approx 2^{111.5}$, and for XHash24-M31, we get a bound of $D_I \geq d^{3\times 24} \approx 2^{167.2}$.

At this stage, we expect neither the bounds to be tight nor such an algorithm to exist. Thus, we claim that the XHash family is safe against GB-based algebraic attacks. The detailed analysis of the attack is described in Appendix G.

5.2.3 Resultant-based Algebraic Attack

The resultant-based algebraic attack described in [38] is based on a different approach, eliminating variables from the multivariate system through the successive computation of resultants. Essentially, the idea is to compute iterated resultants starting from the initial modeling, allowing variables to be eliminated and eventually producing a univariate polynomial. The authors consider two modeling approaches: the direct modeling, referred to as "forward", and the "start-from-the-middle" (SFTM) modeling. In both cases, new variables are introduced at each output of compositional inverse power map S-boxes, which suggests a suitable elimination path. Following the strategies produces a univariate polynomial in the ideal generated by the polynomial system.

In [39], the authors improved upon the resultant method presented in [38] by introducing a new efficient polynomial reduction technique that exploits fast multivariate multiplication. However, they also borrowed theoretical tools from the *FreeLunch* approach.

The resultant-based attack strategies developed so far work for the round functions where the S-boxes are defined over the base field. In the proposed hash functions, however, a new type of round function is introduced, where the S-boxes are defined over the extension fields. Moreover, both of these attacks introduce new variables into the original system only for those rounds that contain S-boxes in the base field, resulting in additional polynomial equations. Consequently, the system becomes more

complex due to the increased number of variables and equations. Therefore, these attack strategies are not practical in the case of our proposed permutations.

6 Benchmarks Of The XHash Family

In this section, we benchmark XHash8-Goldilocks, XHash12-Goldilocks, XHash16-M31, and XHash24-M31 for SNARK and STARK applications. We compare the R1CS, PLONK, and AIR performance of these functions with Rescue-Prime (RP), RPO, Poseidon, Poseidon2, Anemoi, Monolith-64, Monolith-31, and Griffin.

6.1 SNARK performance

For the SNARK performance, we measure the cost of both R1CS and PLONK representations. Let N be the number of rounds, m be the state size, and C_α be the multiplicative cost of each of the S-boxes in the (F)-, (B)-, and the $(B^{'})$-step *i.e.,* $C_\alpha = \lceil log_2(\alpha) \rceil$.

The FFT algorithm multiplies two polynomials of degree d in $(d\ log_2(d)\ log_2(log_2(d))$ operations [28]. In the $(P3)$ round, the state elements are in the extension field. Therefore, $C_\gamma = \lceil d \cdot log_2(d) \cdot log_2(log_2(d) \cdot log_2(\alpha) \rceil$ is the multiplicative cost of each S-box in the $(P3)$ round, and here d is the degree of the irreducible polynomial

Note that the Marvellous design uses two S-boxes in each round. Therefore, it has $2\ C_\alpha$ multiplications per round. In addition, the XHash family uses an extension field S-box along with two base field S-boxes. Therefore, the total multiplication cost for just S-boxes is $2\ C_\alpha + C_\gamma$.

6.1.1 Rank 1 Constraint System (R1CS)

The cost in R1CS is evaluated by $m \cdot N \cdot C_\alpha$. For (s) bit security level, the formulas to calculate the R1CS cost is given in Table 3.

6.1.2 Permutations over Lagrange bases for Oecumenical Noninteractive arguments of knowledge (PLONK)

Standard PLONK complexity without a custom gate is calculated as $m \cdot N \cdot (C_\alpha + L_C)$ where L_C means linear layer constraint (as constant addition is almost free, so it just includes the MDS constraints). MDS constraints are $m(m-1)$ per round. Therefore, the standard PLONK cost is $N \cdot m \cdot (C_\alpha + (m-1))$. For (s) bit security level, the formulas to caluclate the PLONK representation cost are given in Table 3.

6.2 STARK performance

For the STARK performance, we measure the cost of AIR representation. The parameters used in this setting are the monomial degree (α) of the S-box, state size (m), and number of rounds (N).

Table 3: AIR, R1CS, and PLONK cost formulas of RP, RPO, Poseidon, Poseidon2, Anemoi, Monolith-64, Monolith-31, Griffin, XHash12-Goldilocks, XHash24-M31, XHash8-Goldilocks, and XHash16-M31.

	AIR	R1CS	PLONK
RP	$\alpha \cdot m \cdot N$	$2 \cdot m \cdot N \cdot C_\alpha$	$2 \cdot m \cdot N \cdot (C_\alpha + m - 1)$
RPO	$\alpha \cdot m \cdot N$	$2 \cdot m \cdot N \cdot C_\alpha$	$2 \cdot m \cdot N \cdot (C_\alpha + m - 1)$
Poseidon	$\alpha(m \cdot R_F + R_p)$	$(m \cdot R_F + R_P)C_\alpha$	$m(C_\alpha + m - 1)R_F + (C_\alpha + 2 \cdot m - 2)R_P$
Poseidon2	$\alpha(m \cdot R_F + R_p)$	$(m \cdot R_F + R_P)C_\alpha$	$m(C_\alpha + 3)R_F + (C_\alpha + 2 \cdot m - 1)R_P$
Anemoi	$\alpha \cdot m \cdot N$	$N \cdot (m/2) \cdot (C_\alpha + 2)$	$(m/2)(C_\alpha + 5 + m)N$
Monolith-64	$\alpha(68 \cdot N + 64)$	$N(1027 + m)$	$N(1027 + m^2)$
Monolith-31	$\alpha(71 + m)N$	$N(943 + m)$	$N(943 + m^2)$
Griffin	$\alpha \cdot m \cdot N$	$N(2 \cdot \lfloor log_2(C_\alpha) \rfloor + 2 \cdot hw(\alpha) + 2 \cdot m - 6)$	$4(N+1)(m-1) + N(2 \cdot \lfloor log_2(\alpha) \rfloor + 2 \cdot hw(\alpha) + 4 \cdot m - 11)$
XHash12-Goldilocks	$\alpha \cdot m \cdot N$	$(N/2)(2 \cdot m \cdot C_\alpha + (m/3)\ C_\gamma)$	$(N/2)(2 \cdot m \cdot C_\alpha + 2 \cdot m \cdot (m-1) + (m/3)\ C_\gamma)$
XHash24-M31	$\alpha \cdot m \cdot N$	$(N/2)(2 \cdot m \cdot C_\alpha + (m/3)\ C_\gamma)$	$(N/2)(2 \cdot m \cdot C_\alpha + 2 \cdot m \cdot (m-1) + (m/3)\ C_\gamma)$
XHash8-Goldilocks	$\alpha \cdot m \cdot N$	$(N/2)(m \cdot C_\alpha + 2 \cdot (m/3) \cdot C_\alpha + (m/3)\ C_\gamma)$	$(N/2)(m \cdot C_\alpha + 2 \cdot (m/3)\ C_\alpha + 2 \cdot m \cdot (m-1) + (m/3)\ C_\gamma)$
XHash16-M31	$\alpha \cdot m \cdot N$	$(N/2)(m \cdot C_\alpha + 2 \cdot (m/3) \cdot C_\alpha + (m/3)\ C_\gamma)$	$(N/2)(m \cdot C_\alpha + 2 \cdot (m/3)\ C_\alpha + 2 \cdot m \cdot (m-1) + (m/3)\ C_\gamma)$

Table 4: R1CS, PLONK, and AIR costs of RP, RPO, Poseidon, Poseidon2, Anemoi, Monolith-64, Monolith-31, Griffin, XHash12-Goldilocks, XHash8-Goldilocks, XHash24-M31, and XHash16-M31. The most efficient function in each representation is highlighted in bold font. Rows above the double horizontal line correspond to the Goldilocks prime field, while those below correspond to the Mersenne prime field. A dotted horizontal line separates the results of XHash12-Goldilocks, XHash8-Goldilocks, XHash24-M31, and XHash16-M31 from other hash functions.

	α	m	s	N	R1CS	PLONK	AIR
RP	7	12	128	8	576	2688	672
RPO	7	12	128	7	504	2352	588
Poseidon	7	12	128	(8,22)	354	1894	826
Poseidon2	7	12	128	(8,22)	354	1148	826
Anemoi	7	12	128	10	300	1200	840
Monolith-64	2	12	128	6	6234	7026	944
Griffin	7	12	128	9	252	**863**	756
XHash12-Goldilocks	7	12	128	6	276	1068	**504**
XHash8-Goldilocks	7	12	128	6	**240**	1032	**504**
RP	5	24	124	8	1152	9984	960
RPO	-	-	-	-	-	-	-
Poseidon	5	24	124	(8,22)	642	6070	1070
Poseidon2	5	24	124	(8,22)	642	**2252**	1070
Anemoi	5	24	124	9	540	3456	1080
Monolith-31	2	24	124	6	5802	9114	1140
Griffin	-	-	-	-	-	-	-
XHash24-M31	5	24	124	6	552	3864	**720**
XHash16-M31	5	24	124	6	**480**	3792	**720**

6.2.1 Algebraic Intermediate Representation (AIR)

The total cost of representation in AIR is $\alpha \cdot m \cdot N$. For (s) bit security level, the formulas to calculate the AIR representation cost are given in Table 3.

6.3 Comparison

In this section, we compare our proposed hash functions with other hash functions. The results are presented in Table 4.

6.3.1 Goldilocks prime field

We observe that Griffin has the most efficient PLONK representation among all the hash functions presented. In R1CS representation, XHash8-Goldilocks is the most efficient, followed by Griffin. Finally, in AIR, both XHash12-Goldilocks and XHash8-Goldilocks demonstrate the highest efficiency.

6.3.2 Mersenne prime field

We observe that Poseidon2 has the most efficient PLONK representation, followed by Anemoi. In R1CS representation, XHash16-M31 is the most efficient, followed by Anemoi. Finally, in AIR, both XHash24-M31 and XHash16-M31 are the most efficient functions.

7 Conclusion

This work introduced the XHash family, having efficient permutations for zero-knowledge applications with robust security. XHash's security relies on a detailed analysis of different attack vectors, particularly algebraic attacks. For these, we have introduced a new type of security argument with a clear underlying conjecture, which allows us to provide a strong argument against even the most recent variants of this attack. From the performance point of view, the XHash family is more efficient for STARKs *i.e.,* in AIR representation, than all current candidates, making it the most suitable option for the provers in zero-knowledge systems and the verifiers.

Acknowledgement.

The authors thank the reviewers for their constructive comments and suggestions, which helped improve the quality of this work. This work was supported by the VLAIO Baekeland mandate under project number HBC.2024.0256.

Appendix A Example — Deriving The 16th Primitive Root Of Unity In the Mersenne Prime Field (M31)

As an instructive example, we calculate the 16th primitive root of unity in the M31 field.

$$\begin{aligned} e^{\frac{2\pi\iota}{16}} &= cos(\frac{2\pi}{16}) + \iota\ sin(\frac{2\pi}{16}) \\ &= cos(\frac{\pi}{8}) + \iota\ sin(\frac{\pi}{8}) \\ &= \sqrt{\frac{1+cos(\frac{\pi}{4})}{2}} + \iota\ \sqrt{\frac{1-cos(\frac{\pi}{4})}{2}} \\ &= \sqrt{\frac{1+\frac{1}{\sqrt{2}}}{2}} + \iota\ \sqrt{\frac{1-\frac{1}{\sqrt{2}}}{2}} \end{aligned}$$

$$
\begin{aligned}
&= \sqrt{\frac{1+\frac{\sqrt{2}}{2}}{2}} + \iota \sqrt{\frac{1-\frac{\sqrt{2}}{2}}{2}} \\
&= \frac{\sqrt{2+\sqrt{2}}}{2} + \iota \frac{\sqrt{2-\sqrt{2}}}{2} \\
&= \frac{1181536708}{2} + \iota \frac{190298901}{2}
\end{aligned}
$$

Where the penultimate transition uses the fact that the quadratic residue of 2 in M31 is 65536 (*i.e.,* $65536^2 \equiv 2 \ (2^{31}-1)$). In the last transition, the quadratic residues of 65538 and -65534 are 1181536708 and 190298901, respectively. Therefore, after simplification, the 16th primitive root of unity in M31 is:

$$e^{\frac{2\pi\iota}{16}} = 590768354 + \iota\ (1168891274).$$

Appendix B Round Constants

B.1 Rounds Constants for Goldilocks prime field permutaions

```
[2342829598172340146, 3585263893330295736, 9840230240082666977,
8930383252283221116, 9076965446999079099, 8851640424956205527,
13795256210123942050, 11825621142246630355, 15798698224334589477,
5844823629441006380, 16723513721407100738, 7553779927454059727,
3065233571526493382, 11290886347092283424, 4457745179409929033,
10340116268270170156, 11446102672249306457, 4280088487929830122,
3638548349514208164, 3413589566789825026, 3238156475008297440,
17124867911961820747, 6106297290386804040, 3251644546059598585,
7697225986671614633, 2222645750381940862, 17508968376290848167,
15297157869703884803, 3310390846370734828, 9551963521036179107,
12100949535624197538, 17079632298510657864, 3723022722969087405,
2960372941451252329, 5182679596883580618, 17629472291698154013,
13789413482818461197, 13525933623919715862, 10900746068338302936,
14115722142543630475, 12224905253497942814, 12763357285901125395,
6534697208942003524, 5206441130662846401, 2553244522589328764,
10710143529550705140, 15173778526344634162, 4206406657421837479,
10245854197189092080, 2703691184049121959, 3696368915690284528,
6022429258984978244, 9968507192002585156, 15668599340458821381,
617959246242646209, 12222290572019974284, 8441197522255651229,
6869883264255495476, 16493002747194752541, 12468381749755322861,
5538808075640108815, 15029384424752986502, 17289696665364168522,
8638466901924806786, 12399059635537156445, 7326768668263039281,
3622781875014154966, 1042068094006062695, 13540780133400549302,
14947193178926039403, 14587540371260516909, 8019728827279472234,
10267401276070856569, 10747179188588181725, 3837055733227019942,
4081302324177671683, 12836154269049375651, 16374447815452960497,
```

804349192070481435, 4029696078488101528, 1937977348051658235,
804096869528548409, 2947370451308398175, 13535916795517422040,
14236188297302403096, 11783196411574238718, 12953332832705464415,
6712942163197025303, 18168682629967648388, 9044326941982632044,
10572461259095208495, 1534036430854528225, 16629151300772572414,
9387006153095567081, 2681888846780021436, 5675011224831581684,
13290876204454877035, 15560723802551418880, 4771376310669118226,
7937355908567703290, 16391781726194985199, 5774471253979526977,
12796800676917561249, 14898935087580704726, 16795084623262992974,
17206341218177628307, 4770114598059065658, 18135730884164613275,
6154796404442664154, 10336543321945665538, 5378950101993144329,
357951290449577055, 5999235281098382485, 14861517086920414897,
14203846529650605368, 12130083826895746823, 3494633218231290353,
6662848411348181764, 2674749973971814787, 11097731018346290857,
2496822266963220943, 12493593106156612829, 17363278083961362946,
18044118543810187153, 7368286423019705041, 3747592089113714466,
1684305686329477999, 8892738366515227730, 363623219268330632,
16109809237031891469, 7214420470053760925, 5589715653537328267,
6891246656831393530, 18170102401797445950, 5789072777242629178,
17357096097965470126, 17515122248509400444, 17997471997111927514,
1082942864830439301, 12513030475471320370, 7698424518838843479,
18140339834014372828, 9744985442262519794, 17987388816013276356,
18166229948249353102, 4498457789795538924, 14204481340498505964,
4533154768526089885, 110543307065565321, 18443799820168933235,
10808556019384875093, 13069099844718024569, 8609863783820171159,
6746635872973657290, 10622145201887780164, 13996193929634340804,
3566185013190430704, 1076255841717960668, 3937894421727574612,
1509767048442906881, 5609279873837916221, 11405472203818325775,
15279148788716158354, 4157169672629964784, 5730492364936499651,
13956055803137766235, 2030284646456946954, 14453473274285815887,
14862998711426013856, 5406510559932914449, 12347413302270024455,
8264980827485333775, 6136850589998065760, 16846679576113561967,
4475840705012225475, 12996613392073489215, 12637443450867976128,
16903996676759059044, 1288815879640699957, 5906757139360289586,
13804393494056866629, 1990342694594144314, 1345546955952675626,
8970298160242712459, 952183578480181989, 13341913031784154266,
7970308156793749156, 15117035775965098694, 12773906576911070428,
14745226173451075013, 71143947765308918, 14366572994900304087,
6269750249057660071, 12531669991842054633, 9875306748373034281,
1698441867417982010, 6237012358171997457, 6062484447900221462,
8308967414729808905, 13363533295646760621, 241428211288301878,
17741266195769488837, 9083293686816955114, 3836638078782288556,
13596527755874068418, 4317438944248910353, 13121002865081122170,
3690860471161664361, 14963759917558555193, 2355491684881045314,
87863285418571993, 2209803114424326850, 6176212965948490219,

1462630366643264814, 6015835057058442672, 3697516607449911960,
4058655370768082255, 13887926661108047327, 8139967267436566344,
5920057670650792242, 11406469861205362560, 376517680702811945,
4695839913333085020, 7889038229117537094, 9978198557600246887,
3202744387663105304, 453901179763923004, 8064453460280515127,
2033982665883566089, 2238587847542889211, 2250749351334221046,
4556690243642274770, 13757162726085820278, 2320941266176684054,
271415748838667199, 2437064098582070610, 3917407978099157786,
9086663860914841349, 13214412676193605972, 4482678115422768152,
12351689465619309883, 15943492953494662803, 14895590890176007637,
9557270239536312252, 4166576454740400379, 4662225266884298683,
7939645237543490688, 16110192348205074312, 4691529265695197895,
12900728876521008562, 13254876812300979661, 11179346188591236269]

B.2 Rounds Constants for Mersenne prime field permutaions

[175084324, 307267372, 926126032, 968091831, 685157891, 2105385954,
1172337223, 442111374, 619202169, 1608687569, 401276325, 388976039,
747174524, 900395791, 455481706, 590173634, 460285180, 1470272960,
1563942345, 1398899312, 1846418244, 36836460, 1811927922, 167064228,
1142000631, 1434982414, 252489916, 1557799233, 44644995, 571347645,
2114100107, 1615612652, 1413066402, 1869125653, 860912024, 1777495020,
508781706, 1428126799, 416376872, 1627243238, 497670979, 623439811,
142579728, 1866460309, 1828857899, 943424486, 1308979384, 766782025,
715146265, 1305338204, 51159015, 1587489391, 1181321653, 450830656,
1557015574, 1309301546, 1364577641, 1865151097, 1321152955,
1856328988, 1256788027, 2070028866, 142189526, 997525660, 562303485,
1508364161, 546634038, 1855151308, 414030721, 204461777, 2076592910,
2081781835, 1924095286, 696694645, 1016992245, 329457036, 1746153094,
731129352, 1492879802, 109518645, 1506440475, 100640273, 348659820,
1975620907, 434979487, 1607949717, 477749360, 2108279807, 1872602192,
202188144, 1436324583, 352322420, 1267287222, 1759870960, 1698893287,
1409042830, 745465753, 830790775, 1303358776, 199137519, 379646508,
1448243080, 1839615190, 2001025364, 653435503, 1716903235, 1900980930,
127210052, 390073071, 1658571691, 918791597, 215732434, 1641205084,
64509353, 717938853, 1752985684, 549192987, 1277187607, 67103433,
1017165561, 1676100448, 873408598, 846744900, 590232860, 1007269234,
818506088, 397365975, 1627556280, 581304394, 1271111567, 1283713593,
1322890807, 1217915758, 52786324, 1589034864, 868215969, 1806826068,
1096088086, 1008187268, 991073257, 363192403, 156626590, 1864304269,
1797233804, 1592167597, 1870691635, 2116736597, 849359517, 146393037,
1368837761, 1242565219, 703324792, 1936640162, 314274059, 1938208790,
1311570627, 408019024, 488941726, 901158511, 975722452, 245478744,
1081502805, 346944102, 2071265418, 1734880865, 2040503850, 1001865044,
1112077629, 2127315813, 333970284, 576756426, 1421324759, 1034309564,
744510717, 961397525, 754415630, 1300959581, 581645912, 2106918608,

2110129877, 621267186, 1695312995, 609661509, 1913309112, 524300388,
1422632796, 1563605034, 1108753047, 25572393, 1300775045, 523497505,
835791762, 1768040971, 544480237, 388157441, 402171746, 794892581,
1182705154, 49930297, 129243944, 898441973, 429535588, 832127270,
546908747, 756292413, 686066810, 1544391099, 507384091, 625744584,
1233142395, 319177062, 1456409083, 1253953699, 818812123, 1094995825,
1400859865, 777162622, 1879784377, 714561023, 241470607, 1150358400,
868674491, 175364440, 1327652105, 580586163, 612569861, 956760481,
1756883627, 1101943429, 1895088739, 1933031694, 96732478, 224374088,
1381921877, 1325469080, 1889000316, 240001382, 1593956508, 1823462840,
141557806, 530616713, 1907165223, 62079523, 1848686076, 1295897728,
2114545896, 1693023412, 1742672882, 850732884, 196076552, 1821815249,
1719176962, 1889769026, 1061219948, 2025865748, 1939508721, 310849717,
457781360, 517901710, 65968871, 2142772976, 44714611, 101341948,
1253176252, 1930140759, 1799155961, 556217344, 2014050480, 1100303935,
465058938, 511760837, 919812381, 534265201, 908812575, 1963467299,
1628330602, 792487704, 1521690338, 417067392, 1577632259, 1903162100,
1543112201, 1593989960, 2130901602, 2065951024, 2100295115,
1076992712, 1775119720, 221638, 376032096, 1163943790, 1873812351,
375537131, 1957308375, 1069952773, 614213758, 1605486744, 826369864,
1502009496, 659760721, 548669432, 1335550272, 356452350, 675054034,
1993131373, 531587486, 789573063, 1293440420, 1846683787, 1973045453,
768581688, 2002516043, 176434293, 35489330, 821062873, 2030650085,
415910390, 1579605726, 908199138, 1233034527, 1269179452, 80243926,
840306158, 745618431, 209711912, 1828092214, 1665839141, 2134136721,
322680959, 423515746, 47676516, 584020357, 696915461, 1661795164,
2078707993, 584661952, 819846299, 178937197, 1077415206, 1242466361,
1672876753, 1473607717, 1726777726, 644234821, 1262513151, 899912102,
101933161, 1586394069, 727328391, 374984230, 1329273882, 1147628599,
408096360, 1131592455, 1676005029, 1755004107, 1185832027, 1350545055,
234313183, 355535981, 238746508, 813087134, 1623705969, 191472601,
1778703463, 1975849135, 868764694, 366997281, 292403184, 128442573,
1236216701, 666390358, 171867585, 699865044, 1892219207, 349907584,
178565821, 1631281908, 1869158961, 1167585085, 1751361316, 1810451152,
1722258621, 607370421, 1385259181, 852241363, 241792223, 1915594678,
747912505, 1061992187, 678259751, 883281053, 2077909673, 1171481040,
1259386471, 330158658, 899226172, 1120707192, 1291422605, 449005100,
407642177, 1598934822, 472095767, 238981193, 754266611, 724775445,
561137592, 592730614, 510534042, 1435373010, 287991094, 2130283411,
340860454, 1585669652, 179260010, 954133253, 652560445, 1406248378,
539102991, 534552243, 1189488938, 521087116, 611770989, 1646782070,
1539056906, 7599192, 1699865995, 649906382, 1682111335, 1156311902,
225372763, 1906892131, 1665305577, 761480569, 149073325, 1236756677,
293122269, 1540522128, 913784661, 683283264, 1349827183, 302210927,
499701005, 752368547, 1117252871, 1642552701, 1240405350, 1711703709,

```
1345225300, 36959573, 753313677, 2140985386, 933254780, 587810324,
1096977153, 695098010, 1119833811, 1167568020, 203344362, 1933501599,
525914421, 423784548, 1493400934, 1250606839, 1640607559, 878417286,
2100328023, 1685128921, 495027371, 114619005, 137126815, 1022247997,
369347858, 1069621656, 561463310, 233344007, 2079249531, 233542204,
2020769996, 373955554, 1407923718, 310196918, 308842651, 1757125438,
811364578, 1504528972, 1244302447, 1512031330, 1902963598, 1508403531,
356407202, 171711516, 1246960371, 1119845703, 867812005, 2024394375,
1233055993, 1048805681, 305973465, 575344339, 1306988127, 579259204,
1448192336, 291836854]
```

Appendix C 24 x 24 MDS matrix

We recall Theorem 2 from [34]:

Theorem 2 ([34]) *Let $\mathbb{F}_{\mathbb{M}}$ be the Mersenne prime field with modulus $M = 2^{31} - 1$, choose message size $N = 2^n$ with $1 \leq n \leq 30$, and any non-zero λ from $\mathbb{F}_{\mathbb{M}}$. Consider the systematic code over the alphabet $\mathbb{F}_{\mathbb{M}}$ with word size $2 \times N$ and circulant systematic encoding matrix $A = (a_{j,k})$; $0 \leq j, k \leq (N-1)$ defined by*

$$a_{j,k} = \frac{1}{N} \cdot \left[\lambda - \left[\frac{Im(\tau^{1+2(k-j)}}{1 - Re(\tau^{1+2(k-j)})} \right] \right] \tag{C1}$$

where τ is the $(2 \times N)^{th}$ primitive root of unity in the complex extension $\mathbb{C}(\mathbb{F}_{\mathbb{M}})$. Then, independent of the choice of λ, the distance of the code is at least N. In addition, if λ is such that $(\frac{1+i\cdot\lambda}{1-i\cdot\lambda})^{2\cdot N} \neq 1$, then the code has distance $(N+1)$ and thus is maximum distance separable.

C.1 Generation of MDS matrix.

According to Theorem 2, the first possible circulant matrix for a 24-element state is of size 32×32. To find this matrix, we use the 64th primitive root of unity in $\mathbb{C}(\mathbb{F}_M)$, which is given in Equation C2. It can be calculated as in Appendix A.

$$\tau = \frac{\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2}}}}}{2} + \iota \frac{\sqrt{2 - \sqrt{2 + \sqrt{2 + \sqrt{2}}}}}{2} \tag{C2}$$

To represent Equation C2 in $\mathbb{C}(\mathbb{F}_{\mathbb{M}})$, first we calculate the quadratic residues of each term. After simplification, τ in $\mathbb{C}(\mathbb{F}_{\mathbb{M}})$ is $456695729 + \iota(1567857810)$. The first row of the 32×32 circulant matrix is given below, which is generated by calculating Equation C1 using τ. However, to get 24×24 the MDS matrix, we take a submatrix of the circulant matrix having the first 24 rows and 24 columns.

```
[185870542, 2144994796, 1696461115, 215190769, 930115258, 766567118,
2003379079, 1770558586, 1779722644, 434368282, 289154277, 1979813463,
```

```
1436360233, 1342944808, 63026005, 903393155, 1512525948, 105409451,
1072974295, 979558870, 436105640, 2126764826, 1981550821, 636196459,
645360517, 412540024, 1649351985, 1485803845, 53244687, 719457988,
270924307, 82564914].
```

Appendix D Polynomial Representation of π_2 in the Goldilocks Prime Field $\mathbb{F}_G$

Let $(x_0, x_1, x_2) \in \mathbb{F}_{G^3}$ and $(y_0, y_1, y_2) \in \mathbb{F}_{G^3}$ such that $\pi_2(x_0, x_1, x_2) = (y_0, y_1, y_2)$; then

$$\begin{aligned} y_0 = {} & x_0^7 + 35x_0^4x_1^3 + 21x_0^2x_1^5 + 7x_0x_1^6 + x_1^7 + 42x_0^5x_1x_2 + 140x_0^3x_1^3x_2 + 105x_0^2x_1^4x_2 \\ & + 42x_0x_1^5x_2 + 14x_1^6x_2 + 105x_0^4x_1x_2^2 + 210x_0^3x_1^2x_2^2 + 210x_0^2x_1^3x_2^2 + 210x_0x_1^4x_2^2 + 42x_1^5x_2^2 \\ & + 35x_0^4x_2^3 + 140x_0^3x_1x_2^3 + 420x_0^2x_1^2x_2^3 + 280x_0x_1^3x_2^3 + 105x_1^4x_2^3 + 70x_0^3x_2^4 + 210x_0^2x_1x_2^4 \\ & + 315x_0x_1^2x_2^4 + 140x_1^3x_2^4 + 63x_0^2x_2^5 + 168x_0x_1x_2^5 + 105x_1^2x_2^5 + 35x_0x_2^6 + 49x_1x_2^6 + 9x_2^7, \end{aligned}$$

$$\begin{aligned} y_1 = {} & 7x_0^6x_1 + 35x_0^4x_1^3 + 35x_0^3x_1^4 + 21x_0^2x_1^5 + 14x_0x_1^6 + 2x_1^7 + 42x_0^5x_1x_2 + 105x_0^4x_1^2x_2 \\ & + 140x_0^3x_1^3x_2 + 210x_0^2x_1^4x_2 + 84x_0x_1^5x_2 + 21x_1^6x_2 + 21x_0^5x_2^2 + 105x_0^4x_1x_2^2 \\ & + 420x_0^3x_1^2x_2^2 + 420x_0^2x_1^3x_2^2 + 315x_0x_1^4x_2^2 + 84x_1^5x_2^2 + 70x_0^4x_2^3 + 280x_0^3x_1x_2^3 \\ & + 630x_0^2x_1^2x_2^3 + 560x_0x_1^3x_2^3 + 175x_1^4x_2^3 + 105x_0^3x_2^4 + 420x_0^2x_1x_2^4 + 525x_0x_1^2x_2^4 \\ & + 245x_1^3x_2^4 + 105x_0^2x_2^5 + 294x_0x_1x_2^5 + 189x_1^2x_2^5 + 63x_0x_2^6 + 84x_1x_2^6 + 16x_2^7, \end{aligned}$$

$$\begin{aligned} y_2 = {} & 21x_0^5x_1^2 + 35x_0^3x_1^4 + 21x_0^2x_1^5 + 7x_0x_1^6 + 2x_1^7 + 7x_0^6x_2 + 105x_0^4x_1^2x_2 + 140x_0^3x_1^3x_2 \\ & + 105x_0^2x_1^4x_2 + 84x_0x_1^5x_2 + 14x_1^6x_2 + 21x_0^5x_2^2 + 105x_0^4x_1x_2^2 + 210x_0^3x_1^2x_2^2 \\ & + 420x_0^2x_1^3x_2^2 + 210x_0x_1^4x_2^2 + 63x_1^5x_2^2 + 35x_0^4x_2^3 + 280x_0^3x_1x_2^3 + 420x_0^2x_1^2x_2^3 \\ & + 420x_0x_1^3x_2^3 + 140x_1^4x_2^3 + 70x_0^3x_2^4 + 315x_0^2x_1x_2^4 + 420x_0x_1^2x_2^4 + 175x_1^3x_2^4 + 84x_0^2x_2^5 \\ & + 210x_0x_1x_2^5 + 147x_1^2x_2^5 + 49x_0x_2^6 + 63x_1x_2^6 + 12x_2^7. \end{aligned}$$

Similarly

$$\pi_2(x_0^{(2G-1)/7}, x_1, x_2^{(2G-1)/7}) = (y_0, y_1, y_2)$$

gives

$$
\begin{aligned}
y_0 = {} & x_0^{(2G-1)/7} + 35x_0^{(8G-4)/7}x_1^3 + 21x_0^{(4G-2)/7}x_1^5 + 7x_0^{(2G-1)/7}x_1^6 + x_1^7 \\
& + 42x_0^{(10G-5)/7}x_1x_2^{(2G-1)/7} + 140x_0^{(6G-3)/7}x_1^3x_2^{(2G-1)/7} + 105x_0^{(4G-2)/7}x_1^4x_2^{(2G-1)/7} \\
& + 42x_0^{(2G-1)/7}x_1^5x_2^{(2G-1)/7} + 14x_1^6x_2^{(2G-1)/7} + 105x_0^{(8G-4)/7}x_1x_2^{(4G-2)/7} \\
& + 210x_0^{(6G-3)/7}x_1^2x_2^{(4G-2)/7} + 210x_0^{(4G-2)/7}x_1^3x_2^{(4G-2)/7} + 210x_0^{(2G-1)/7}x_1^4x_2^{(4G-2)/7} \\
& + 42x_1^5x_2^{(4G-2)/7} + 35x_0^{(8G-4)/7}x_2^{(6G-3)/7} + 140x_0^{(6G-3)/7}x_1x_2^{(6G-3)/7} \\
& + 420x_0^{(4G-2)/7}x_1^2x_2^{(6G-3)/7} + 280x_0^{(2G-1)/7}x_1^3x_2^{(6G-3)/7} + 105x_1^4x_2^{(6G-3)/7} \\
& + 70x_0^{(6G-3)/7}x_2^{(8G-4)/7} + 210x_0^{(4G-2)/7}x_1x_2^{(8G-4)/7} + 315x_0^{(2G-1)/7}x_1^2x_2^{(8G-4)/7} \\
& + 140x_1^3x_2^{(8G-4)/7} + 63x_0^{(4G-2)/7}x_2^{(10G-5)/7} + 168x_0^{(2G-1)/7}x_1x_2^{(10G-5)/7} \\
& + 105x_1^2x_2^{(10G-5)/7} + 35x_0^{(2G-1)/7}x_2^{(12G-6)/7} + 49x_1x_2^{(12G-6)/7} + 9x_2^{(2G-1)},
\end{aligned}
$$

$$
\begin{aligned}
y_1 = {} & 7x_0^{(12G-6)/7}x_1 + 35x_0^{(8G-4)/7}x_1^3 + 35x_0^{(6G-3)/7}x_1^4 + 21x_0^{(4G-2)/7}x_1^5 \\
& + 14x_0^{(2G-1)/7}x_1^6 + 2x_1^7 + 42x_0^{(10G-5)/7}x_1x_2^{(2G-1)/7} + 105x_0^{(8G-4)/7}x_1^2x_2^{(2G-1)/7} \\
& + 140x_0^{(6G-3)/7}x_1^3x_2^{(2G-1)/7} + 210x_0^{(4G-2)/7}x_1^4x_2^{(2G-1)/7} + 84x_0^{(2G-1)/7}x_1^5x_2^{(2G-1)/7} \\
& + 21x_1^6x_2^{(2G-1)/7} + 21x_0^{(10G-5)/7}x_2^{(4G-2)/7} + 105x_0^{(8G-4)/7}x_1x_2^{(4G-2)/7} \\
& + 420x_0^{(6G-3)/7}x_1^2x_2^{(4G-2)/7} + 420x_0^{(4G-2)/7}x_1^3x_2^{(4G-2)/7} + 315x_0^{(2G-1)/7}x_1^4x_2^{(4G-2)/7} \\
& + 84x_1^5x_2^{(4G-2)/7} + 70x_0^{(8G-4)/7}x_2^{(6G-3)/7} + 280x_0^{(6G-3)/7}x_1x_2^{(6G-3)/7} \\
& + 630x_0^{(4G-2)/7}x_1^2x_2^{(6G-3)/7} + 560x_0^{(2G-1)/7}x_1^3x_2^{(6G-3)/7} + 175x_1^4x_2^{(6G-3)/7} \\
& + 105x_0^{(6G-3)/7}x_2^{(8G-4)/7} + 420x_0^{(4G-2)/7}x_1x_2^{(8G-4)/7} + 525x_0^{(2G-1)/7}x_1^2x_2^{(8G-4)/7} \\
& + 245x_1^3x_2^{(8G-4)/7} + 105x_0^{(4G-2)/7}x_2^{(10G-5)/7} + 294x_0^{(2G-1)/7}x_1x_2^{(10G-5)/7} \\
& + 189x_1^2x_2^{(10G-5)/7} + 63x_0^{(2G-1)/7}x_2^{(12G-6)/7} + 84x_1x_2^{(12G-6)/7} + 16x_2^{(2G-1)},
\end{aligned}
$$

$$
\begin{aligned}
y_2 = {} & 21x_0^{(10G-5)/7}x_1^2 + 35x_0^{(6G-3)/7}x_1^4 + 21x_0^{(4G-2)/7}x_1^5 + 7x_0^{(2G-1)/7}x_1^6 + 2x_1^7 \\
& + 7x_0^{(12G-6)/7}x_2^{(2G-1)/7} + 105x_0^{(8G-4)/7}x_1^2x_2^{(2G-1)/7} + 140x_0^{(6G-3)/7}x_1^3x_2^{(2G-1)/7} \\
& + 105x_0^{(4G-2)/7}x_1^4x_2^{(2G-1)/7} + 84x_0^{(2G-1)/7}x_1^5x_2^{(2G-1)/7} + 14x_1^6x_2^{(2G-1)/7} \\
& + 21x_0^{(10G-5)/7}x_2^{(4G-2)/7} + 105x_0^{(8G-4)/7}x_1x_2^{(4G-2)/7} + 210x_0^{(6G-3)/7}x_1^2x_2^{(4G-2)/7} \\
& + 420x_0^{(4G-2)/7}x_1^3x_2^{(4G-2)/7} + 210x_0^{(2G-1)/7}x_1^4x_2^{(4G-2)/7} + 63x_1^5x_2^{(4G-2)/7} \\
& + 35x_0^{(8G-4)/7}x_2^{(6G-3)/7} + 280x_0^{(6G-3)/7}x_1x_2^{(6G-3)/7} + 420x_0^{(4G-2)/7}x_1^2x_2^{(6G-3)/7} \\
& + 420x_0^{(2G-1)/7}x_1^3x_2^{(6G-3)/7} + 140x_1^4x_2^{(6G-3)/7} + 70x_0^{(6G-3)/7}x_2^{(8G-4)/7} \\
& + 315x_0^{(4G-2)/7}x_1x_2^{(8G-4)/7} + 420x_0^{(2G-1)/7}x_1^2x_2^{(8G-4)/7} + 175x_1^3x_2^{(8G-4)/7} \\
& + 84x_0^{(4G-2)/7}x_2^{(10G-5)/7} + 210x_0^{(2G-1)/7}x_1x_2^{(10G-5)/7} + 147x_1^2x_2^{(10G-5)/7} \\
& + 49x_0^{(2G-1)/7}x_2^{(12G-6)/7} + 63x_1x_2^{(12G-6)/7} + 12x_2^{(2G-1)}.
\end{aligned}
$$

Appendix E Polynomial Representation of π_2 in the Mersenne Prime Field $\mathbb{F}_M$

Let $(x_0, x_1, x_2) \in \mathbb{F}_{M^3}$ and $(y_0, y_1, y_2) \in \mathbb{F}_{M^3}$ such that $\pi_2(x_0, x_1, x_2) = (y_0, y_1, y_2)$;

$$\begin{aligned} y_0 = x_0^5 + 2147483587x_0^3x_1x_2 + 2147483627x_0^2x_1^3 + 40x_0^2x_2^3 \\ + 120x_0x_1^2x_2^2 + 20x_1^4x_2 + 2147483607x_1x_2^4, \end{aligned}$$

$$\begin{aligned} y_1 = 5x_0^4x_1 + 2147483627x_0^3x_2^2 + 2147483587x_0^2x_1^2x_2 + 2147483637x_0x_1^4 \\ + 40x_0x_1x_2^3 + 40x_1^3x_2^2 + 2147483639x_2^5, \end{aligned}$$

$$\begin{aligned} y_2 = 5x_0^4x_2 + 10x_0^3x_1^2 + 2147483587x_0^2x_1x_2^2 + 2147483587x_0x_1^3x_2^3 \\ + 20x_0x_2^4 + 2147483645x_1^5 + 40x_1^2x_2^3. \end{aligned}$$

Similarly

$$\pi_2(x_0^{(2M-1)/5}, x_1, x_2^{(2M-1)/5}) = (y_0, y_1, y_2)$$

gives

$$\begin{aligned} y_0 = x_0^{(10M-5)/5} + 2147483587x_0^{(6M-3)/5}x_1x_2^{(2M-1)/5} + 2147483627x_0^{(4M-2)/5}x_1^3 \\ + 40x_0^{(4M-2)/5}x_2^{(6M-3)/5} + 120x_0^{(2M-1)/5}x_1^2x_2^{(4M-2)/5} \\ + 20x_1^4x_2^{(2M-1)/5} + 2147483607x_1x_2^{(8M-4)/5}, \end{aligned}$$

$$\begin{aligned} y_1 = 5x_0^{(8M-4)/5}x_1 + 2147483627x_0^{(6M-3)/5}x_2^{(4M-2)/5} \\ + 2147483587x_0^{(4M-2)/5}x_1^2x_2^{(2M-1)/5} + 2147483637x_0^{(2M-1)/5}x_1^4 \\ + 40x_0^{(2M-1)/5}x_1x_2^{(6M-3)/5} + 40x_1^3x_2^{(4M-2)/5} + 2147483639x_2^{(10M-5)/5}, \end{aligned}$$

$$\begin{aligned} y_2 = 5x_0^{(8M-4)/5}x_2^{(2M-1)/5} + 10x_0^{(6M-3)/5}x_1^2 + 2147483587x_0^{(4M-2)/5}x_1x_2^{(4M-2)/5} \\ + 2147483587x_0^{(2M-1)/5}x_1^3x_2^{(2M-1)/5} + 20x_0^{(2M-1)/5}x_2^{(8M-4)/5} \\ + 2147483645x_1^5 + 40x_1^2x_2^{(6M-3)/5}. \end{aligned}$$

Appendix F Gröbner Basis Resistance

The computation of Gröbner basis is based on each type of round. Recalling that the first type of round contains two times constant additions, two times an MDS matrix, π_0, and π_1. The second type of round contains constant addition and π_2. In our experiments, we used a toy instance with state size $m = 3$, (*rate*) $r = 1$, and (*capacity*) $c = 2$.

Polynomial description. The input is denoted by:

$$X_0 = (X_0[1], \ldots, X_0[r], 0, \ldots, 0)\,,$$

the state after the step i by

$$X_i = (X_i[1], \ldots, X_i[m])\,,$$

and the output by

$$Y = (H[1], \ldots, H[r], Y[r+1], \ldots, Y[m])$$

where $H[i]$ is the ith output of the hash function. The MDS matrix is denoted by M and constants used at step i are denoted by

$$K_i = (K_i[1], \ldots, K_i[m])\,, K_i' = (K_i'[1], \ldots, K_i'[m])\,.$$

In the case of the XHash12 and XHash24 permutations, the ith basic step is modeled as follows:

$$\left(\sum_{k=0}^{m-1} M[j,k] \left(\sum_{k=0}^{m-1} M[j,k] \left[S_{i-1}[j] + K_i[j]\right]\right)^{\alpha}\right) + K_i^{'}[j] - S_i^{\alpha}[j] = 0$$

In the case of the XHash8 and XHash16 permutations, the ith basic step is modeled as follows:

$$\left(\sum_{k=0}^{m-1} M[j,k] \left(\sum_{k=0}^{m-1} M[j,k] \left[S_{i-1}[j] + K_i[j]\right]\right)^{\alpha}\right) + K_i^{'}[j] - S_i^{\alpha}[j] = 0 \quad j \not\equiv 1 \pmod 3$$

$$\left(\sum_{k=0}^{m-1} M[j,k] \left(\sum_{k=0}^{m-1} M[j,k] \left[S_{i-1}[j] + K_i[j]\right]\right)^{\alpha}\right) + K_i^{'}[j] - S_i[j] = 0 \quad j \equiv 1 \pmod 3$$

The ith extended step is modeled as:

$$(S_i[j] + K_{i+1}[j]) - \pi_{2,\beta} \equiv 0, \qquad \beta = j \mod 3$$

The results of the Gröbner basis algorithm for XHash12 and XHash24 with $m = 3, r = 1, p = 65519$ are described in Table F1 and Table F2.

Introducing partial layers in XHash8 and XHash16 results, as expected, in a smaller or equal solving degree. Interestingly, however, both the actual running time and the memory consumption increase. These experiments with toy parameters are described in Table F3 and Table F4.

Table F1: Experimental results for finding a Gröbner basis in degrevlex order for XHash12

N	$\mathcal{V}$	SD	D_{reg}	Complexity	Time(s)	Mem(MB)
1	3	7	19	13.81	0.01	32
2	6	12	37	28.36	46	224
3	9	14	55	39.28	2921	11514

Table F2: Experimental results for finding a Gröbner basis in degrevlex order for XHash24

N	$\mathcal{V}$	SD	D_{reg}	Complexity	Time(s)	Mem(MB)
1	3	5	13	11.62	0.01	32
2	6	9	25	24.58	1.23	120
3	9	10	37	32.99	39.55	320

Table F3: Experimental results of finding a Gröbner basis in the degrevlex order for XHash8

N	$\mathcal{V}$	SD	D_{reg}	Complexity	Time(s)	Mem(MB)
1	3	7	19	13.81	0.0	32
2	6	12	37	28.36	55	288
3	9	14	55	39.28	3608	14817

The total complexity of computing the Gröbner basis in degrevlex order for the hash function with N rounds, state size m, and rate r is:

$$\mathcal{C}_{gb} \geq \binom{\mathcal{V} + d_{sol}}{d_{sol}}^2. \tag{F3}$$

By taking a very conservative approximation and assuming the solving degree remains 14 after the third round, then the complexity of Goldilocks prime based instances is lower bounded by

$$\mathcal{C}_{gb} \geq 2^{104.02}.$$

While complexity of Mersenne prime-based instances is lower bounded by

$$\mathcal{C}_{gb} \geq 2^{100.9}.$$

Table F4: Experimental results for finding a Gröbner basis in degrevlex order for XHash16

N	$\mathcal{V}$	SD	D_{reg}	Complexity	Time(s)	Mem(MB)
1	3	5	13	11.62	0.04	32
2	6	8	25	23.10	1.43	309
3	9	10	37	32.99	395	826

Appendix G Detailed Multivariate Algebraic Cryptanalysis

G.1 Modeling XHash

In what follows, letters x, y denote variables, and Greek letters denote multivariate polynomials in a prime field. The inner operations of XHash are denoted with uppercase Latin letters.

G.1.1 General Approach.

For each permutation of XHash, we use a similar approach. In order to ensure that the system is expected to have a single solution (or at most a few), we need to restrict the input to a vector space of dimension r, r being the *rate*. The simplest approach consists simply of starting with variables $x_0, \cdots, x_{r-1}$ for the input of the permutation, setting the others to 0.

We then construct polynomial constraints to ensure that the internal state at the output of the permutation is indeed the image of the initial state after the relevant operations. Passing through low-degree operations is straightforward, as we simply update the polynomials by composing them with those operations. For the (partial) layer of the 7th and 5th roots, we proceed differently: we introduce variables y_j^i just after each 7th and 5th root, where i denotes the round index and j the state index.

First Round.

For the first round, we simply need to encode that the inputs of each 7th and 5th root are obtained by applying two MDS layers and a layer of small monomials (along with the appropriate round constants). We deduce 12 equations (resp. 8) for XHash12 (resp. XHash8) and 24 equations (resp. 16) for XHash24 (resp. XHash16) of the following shapes:

$$(y_j^1)^7 \;=\; C_1 + L_j\left(S\left(L\left(C_0 + \vec{x}\right)\right)\right) \;=\; \alpha_j(x)\ .$$

$$(y_j^1)^5 \;=\; C_1 + L_j\left(S\left(L\left(C_0 + \vec{x}\right)\right)\right) \;=\; \alpha_j(x)\ .$$

To look for solutions to the CICO problem — or indeed for preimages, we need to fix the *capacity* words to 0. To this end, we consider that $\vec{x} = (x_0, ..., x_{r-1}, 0, ..., 0)$.

Middle Rounds.

We get equations of the following types for the middle rounds of XHash12 and XHash24:

$$(y_j^{i+1})^7 \;=\; C_i + L_j\left(S\left(L\left(C_i + T(y^i)\right)\right)\right) \;=\; \mu_i(y^i)\;. \tag{G4}$$

$$(y_j^{i+1})^5 \;=\; C_i + L_j\left(S\left(L\left(C_i + T(y^i)\right)\right)\right) \;=\; \mu_i(y^i)\;. \tag{G5}$$

Since the layer of small S-boxes (S) and the layer of big S-boxes (T) are both of degree 7 and 5 in the Goldilocks and Mersenne prime fields, and since the MDS layer L ensures that all outputs depend on all inputs, we have that μ_i is always of degree $7^2 = 49$ in the Goldilocks prime field and $5^2 = 25$ in the Mersenne prime field.

For XHash8, the situation is similar except that we introduce fewer such variables (only 8 per round) and that the polynomial constraints take as input the variables from all the previous rounds, not only the immediate predecessor. It complicates writing a closed formula for these constraints, but they can be generated recursively starting from the first one with a computer program that updates the expression of the polynomials corresponding to the "identity" boxes in the high degree layer and keeps those in memory. We then get equations of the form

$$(y_j^{i+1})^7 \;=\; \mu'_i(y^i, y^{i-1}, ..., y^1, x)\;. \tag{G6}$$

As for μ_i, the polynomials μ'_i are of degree 49 in y^i. However, their degree in the previous variables is higher since each of those went through operations of degree 49 during each round.

For XHash16, we get the equations of the form

$$(y_j^{i+1})^5 \;=\; \mu'_i(y^i, y^{i-1}, ..., y^1, x)\;. \tag{G7}$$

As for μ_i, the polynomials μ'_i are of degree 25 in y^i. However, their degree in the previous variables is higher since each of those went through operations of degree 25 during each round.

Final Rounds.

For the final round, we do not introduce any new variable. However, we add one equation per state element whose value needs to be set to a specific value. Essentially, we get several affine combinations of the image under the big S-box of the y_j^r and (for XHash8 and XHash16) some complex polynomials in all the previous variables, which we denote ω_i.

Bypassing Rounds.

A natural attack angle at this stage is to try and simplify the system using the knowledge of its structure, a trick usually achieved by carefully tracking the propagation of some affine spaces. This was applied with some success to several SPNs in [28], and even more so against both Griffin and Arion [22] due to the specifics of their non-linear

layer — in particular, the fact that the non-linear layer of Griffin is affine over large affine spaces.

However, we could not find such heuristics here. In the first round, the first MDS layer restricts the control needed to apply the technique from [28]. The fact that the non-linear layer is of a very high degree everywhere[6] then prevents the applicability of the technique from [22]. The monomial-based non-linear layer could have potentially led to the existence of chains of subspaces, as in [40], but the dense and structure-less round constants are an effective countermeasure against it.

Furthermore, the use of the monomial over cubic extension fields in the big S-box layer makes it complicated to play with the algebraic representation of several rounds by tightly interweaving its 3 input variables.

Overall, while the threat from "free" rounds might a priori be pressing due to the low total number of rounds, it seems like the built-in countermeasures make such threats impractical.

G.2 A Specific Monomial Ordering

In the end, we have two types of variables: the x_j, which correspond to the input, and the y_j^i, which are grouped in layers corresponding to the round at which they are introduced. We order them using a weighted *Grevlex* order, whereby variables x_j have weight 1, and a variable y_j^i (introduced during round i) has weight α^{2i}.

Our system of equations is of the form

$$\begin{cases} 0 = (y_0^0)^\alpha - \alpha_0(\vec{x}) \\ \quad\quad ... \\ 0 = (y_{\ell-1}^0)^\alpha - \alpha_{11}(\vec{x}) \\ 0 = (y_0^1)^\alpha - \mu_0\left(\vec{x}, \vec{y^0}\right) \\ \quad\quad ... \\ 0 = (y_{\ell-1}^{N_r})^\alpha - \mu_0\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r-1}}\right) \\ 0 = \omega_0\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r}}\right) \\ \quad\quad ... \\ 0 = \omega_{m-\ell}\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r}}\right) \ , \end{cases} \tag{G8}$$

where $\alpha = 7$ in the Goldilocks prime field and $\alpha = 5$ in the Mersenne prime field. Moreover, $\ell = 8$ for XHash8, $\ell = 12$ for XHash12, $\ell = 16$ for XHash16, and $\ell = 24$ for XHash24.

By construction, our custom ordering ensures that the leading monomials in all the starting and middle equations are different variables (namely, all the y_j^i). For XHash8, in the case where we only try to force one 0 in the input and in the output, this is exactly a FreeLunch system: the leading monomial in the last equation is of the form $x_0^{49(r+1)}$, meaning that it is yet another variable and that the whole system is a

[6] Except for XHash8 and XHash16, where some inverse monomials are not applied, but this is of no consequence here.

Gröbner basis out-of-the-box. Similarly, for XHash16, we have a Freelunch system in which the leading polynomial of the last equation is of the form $x_0^{25(r+1)}$.

Unfortunately, it is not the case for both XHash12 and XHash24, or when the number of attacked output blocks (and thus the number of variables in $\vec{x}$) is at least equal to 2. Still, we do have that the leading monomials in all but the final equations are pairwise distinct.

G.2.1 On XHash12 and XHash24

In both XHash12 and XHash24, the input variables do not play a specific role since the inverse monomial layers are full. Indeed, we lose the ability to force the input variables to play a specific role in the last round and thus cannot have the x_j be the only variables in the leading monomials in any equation.

We thus considered alternative strategies to build the system of equations for both and, in particular, to build the weights so that they decrease with the number of rounds. Thus forcing the x_i to be the leading terms in the first round, the $\vec{y^i}$ to be the leading terms in the second round, *etc,*. However, the inner workings of the round function prevented us from going further: it is "easy" to force y^d to be the leading term of an equation of the form $y^d - P(y,\vec{x})$, but much more complicated to pick a specific variable intervening in the rather dense polynomial P, and have it be the only variable in the leading monomial of such an equation.

In the end, we use the same encoding for XHash12 as we do for XHash8 and for XHash24 as we do for XHash16, and in fact, they can be built using the same program.

G.3 Experimental Results

It is easy to design (and implement) a system of equations corresponding to an attack against a greatly simplified XHash8, differing from the real one in the following ways.

- Use $x \mapsto x^3$ as a small S-box,
- Use a lower value for p (namely, $p = 2^{16} - 17$) such that $x \mapsto x^3$ is a permutation,
- Decrease the number of branches and consider different patterns for the application of the cubic roots.
- Resize the big S-box to operate on $\mathbb{F}_p^2$ rather than $\mathbb{F}_p^3$; and allow its replacement by a simple (full) layer of branch-wide monomials (to better understand the impact of big S-boxes).

The attached `Python` script does just that. We call ℓ the number of cubic roots applied in parallel in each round (so that $\ell = 8$ for XHash8 for instance), and w the number of both x_i and of final equations.[7]

G.3.1 Confirming the MIDC

Running experiments on a small number of rounds, we have found that the Monotonous Ideal Degree Conjecture held. In fact, a stronger result seems to hold in

[7] We need these to be equal to maintain an ideal of dimension 0.

the case of XHash8: the ideal degree seems equal to

$$\underbrace{d^{\ell r}}_{y_i} \times \underbrace{d^{k \times 2r}}_{x_j},$$

which is $d^{(k-1)2r}$ times bigger than the bound given by the conjecture.

On the Impact of T.

The much denser T operation (operating over $\mathbb{F}_p^3$) does not influence the degree of the ideal. In that sense, it a priori does not provide more resistance against algebraic attacks than a simpler layer of monomials applied on each branch.

However, in practice, it does imply some important properties. First, its greater density means that all the polynomials have an observable higher Hamming weight — a property that gets stronger as the exponent increases. This means that the equations are hard to generate and manipulate, and that targeting the (elimination of) specific terms is likely to be much more complicated. More importantly, making the polynomials denser is important to prevent the algorithm targeting sparse systems from becoming applicable.

Furthermore, by operating on several state elements at once, it prevents an attacker from singling out any of them. Being able to do this is important to select which variables to pick as the leading monomials or to find heuristics to bypass rounds for free. As a consequence, we consider that these layers offer a significant security increase, albeit not one that is directly visible in the ideal degree.

References

[1] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. **2018**, 46 (2018)

[2] Ashur, T., Mahzoun, M., Toprakhisar, D.: Chaghri - A FHE-friendly block cipher. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 139–150 (2022)

[3] Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst meets fluid-SPN: Griffin for zero-knowledge applications. In: Annual International Cryptology Conference, pp. 573–606 (2023). Springer

[4] Bouvier, C., Briaud, P., Chaidos, P., Perrin, L., Salen, R., Velichkov, V., Willems, D.: New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and Jive compression mode. In: Annual International Cryptology Conference, pp. 507–539 (2023). Springer

[5] Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 430–454 (2015). Springer

[6] Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Journal of Cryptology **31**(3), 885–916 (2018)

[7] Méaux, P., Journault, A., Standaert, F.-X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 311–343 (2016). Springer

[8] Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: a cipher with low AND depth and few ANDs per bit. In: Annual International Cryptology Conference, pp. 662–692 (2018). Springer

[9] Hebborn, P., Leander, G.: Dasta – alternative linear layer for Rasta. IACR Transactions on Symmetric Cryptology, 46–86 (2020)

[10] Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., Walch, R.: Pasta: A case for hybrid homomorphic encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(3), 30–73 (2023)

[11] Cid, C., Indrøy, J.P., Raddum, H.: FASTA – A stream cipher for fast FHE evaluation. In: Cryptographers' Track at the RSA Conference, pp. 451–483 (2022). Springer

[12] Ha, J., Kim, S., Lee, B., Lee, J., Son, M.: Rubato: Noisy ciphers for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 581–610 (2022). Springer

[13] Szepieniec, A., Lemmens, A., Sauer, J.F., Threadbare, B., et al.: The Tip5 hash function for recursive starks. Cryptology ePrint Archive (2023)

[14] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 191–219 (2016). Springer

[15] Grassi, L., Khovratovich, D., Schofnegger, M.: Poseidon2: A faster version of the Poseidon hash function. In: International Conference on Cryptology in Africa, pp. 177–203 (2023). Springer

[16] Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations. IACR Transactions on Symmetric Cryptology, 44–83 (2024)

[17] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of

symmetric-key primitives for advanced cryptographic protocols. IACR Transactions on Symmetric Cryptology, 1–45 (2020)

[18] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for Zero-Knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 519–535 (2021)

[19] Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-Prime: A standard specification (SoK). Cryptology ePrint Archive (2020)

[20] Ashur, T., Kindi, A., Meier, W., Szepieniec, A., Threadbare, B.: Rescue-Prime Optimized. Cryptology ePrint Archive (2022)

[21] Roy, A., Steiner, M.J., Trevisani, S.: Arion: Arithmetization-Oriented permutation and hashing from generalized triangular dynamical systems. arXiv preprint arXiv:2303.04639 (2023)

[22] Bariant, A., Boeuf, A., Lemoine, A., Manterola Ayala, I., Øygarden, M., Perrin, L., Raddum, H.: The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In: Annual International Cryptology Conference, pp. 139–173 (2024). Springer

[23] Daemen, J., Rijmen, V.: The Design of Rijndael: AES — The Advanced Encryption Standard. Information Security and Cryptography, vol. 2, pp. 1–238. Springer, Berlin, Heidelberg (2002)

[24] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 181–197 (2008). Springer

[25] Ashur, T., Bhati, A.S.: Generalized Indifferentiable Sponge and its Application to Polygon Miden VM. Cryptology ePrint Archive (2024)

[26] Aumasson, J.-P., Khovratovich, D., Mennink, B., Quine, P.: SAFE: Sponge API for field elements. Cryptology ePrint Archive (2023)

[27] Khovratovich, D., Beltrán, M.M., Mennink, B.: Generic security of the SAFE API and its applications. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 301–327 (2023). Springer

[28] Bariant, A., Bouvier, C., Leurent, G., Perrin, L.: Algebraic attacks against some arithmetization-oriented primitives. IACR Transactions on Symmetric Cryptology, 73–101 (2022)

[29] Faugere, J.-C.: A new efficient algorithm for computing Gröbner basis (F4). Journal of pure and applied algebra **139**(1-3), 61–88 (1999)

[30] Faugere, J.C.: A new efficient algorithm for computing Gröbner basis without

reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, pp. 75–83 (2002)

[31] Faugere, J.-C., Gianni, P., Lazard, D., Mora, T.: Efficient computation of zero-dimensional Gröbner basis by change of ordering. Journal of Symbolic Computation **16**(4), 329–344 (1993)

[32] Tariq, S. https://github.com/3MI-Labs/XHash-Constants.git (2024)

[33] Haböck, U., Schofnegger, M.: Circulant and other efficiently evaluable MDS matrices from Reed-Solomon codes. Unpublished manuscript. Available at https://hackmd.io/@sKYgEqCsSZW5mqQfCGUHvA/SkUsv8qAZg (accessed May 11, 2026) (2026)

[34] Haböck, U.: A note on circulant MDS matrices. Private Correspondence (2024)

[35] Keller, N., Rosemarin, A.: Mind the middle layer: the HADES design strategy revisited. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 35–63 (2021). Springer

[36] Ashur, T., Buschman, T., Mahzoun, M.: Algebraic cryptanalysis of POSEIDON. IACR Cryptol. ePrint Arch. **2023**, 537 (2023)

[37] Sauer, J.F.: Gröbner basis-attacking a tiny sponge. Technical report, Technical report, AS Discrete Mathematics. (2021)

[38] Yang, H.-S., Zheng, Q.-X., Yang, J., Liu, Q.-f., Tang, D.: A new security evaluation method based on resultant for arithmetic-oriented algorithms. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 457–489 (2024). Springer

[39] Bariant, A., Boeuf, A., Briaud, P., Hostettler, M., Øygarden, M., Raddum, H.: Improved resultant attack against arithmetization-oriented primitives. In: Annual International Cryptology Conference, pp. 335–367 (2025). Springer

[40] Boeuf, A., Canteaut, A., Perrin, L.: Propagation of Subspaces in Primitives with Monomial Sboxes: Applications to Rescue and Variants of the AES. IACR Transactions on Symmetric Cryptology **2023**(4), 270–298 (2023)