RainHash2.0: Hardware- and Arithmetization-friendly Hash Function

Jiamin Cui[6], Lorenzo Grassi[3,4], Katharina Koschatko[1], Florian Krieger[1], Shibam Mukherjee[1,2], Christian Rechberger[1,7], Sujoy Sinha Roy[1], Markus Schofnegger[5] and Verena Schröppel[1,7]

[1] Graz University of Technology, Graz, Austria, {firstname.lastname}@tugraz.at
[2] Know Center, Graz, Austria
[3] Eindhoven University of Technology, Eindhoven, Netherlands, l.grassi@tue.nl
[4] Ponos Technology, Zug, Switzerland
[5] [[alloc] init], San Francisco, United States, markus.schofnegger@gmail.com
[6] Shandong University, Qingdao, China, cuijiamin@sdu.edu.cn
[7] TACEO, Graz, Austria

Abstract.
Zero-knowledge (ZK) proof systems have developed rapidly in recent years, with hash functions as one of their central building blocks. Since these often dominate the prover cost, circuit-friendly hash function design has become an active research area. Most hash proposals target prime fields, although recent protocols such as Binius and VOLE-based ZK operate natively over binary extension fields $\mathbb{F}_{2^n}$. These binary field protocols reduce the cost of proving widely used binary and bitwise statements, thereby opening up new design opportunities. At the same time, the demand for ZK applications such as zkRollups is pushing towards performant hardware acceleration, a requirement that recent designs have largely neglected. Hence, a modern ZK hash function should also be efficient in hardware and fast in plain evaluation, to avoid new bottlenecks in non-circuit workloads.
In this paper, we introduce RainHash2.0, a cryptographic permutation that addresses both gaps. RainHash2.0 is natively defined over binary extension fields, making it a natural match for $\mathbb{F}_{2^n}$-based protocols such as Binius and VOLEitH, while being tailored for efficient hardware and competitive plain performance. To achieve this, we exploit new techniques from Binius to horizontally split the round function – arguably a novelty in itself that is particularly effective when finite fields of different sizes are used simultaneously. We implement RainHash2.0 in the Binius and VOLE-based ZK frameworks, comparing it against SHAKE, recent arithmetization-oriented designs, and its direct predecessor RainHash. Across proof size, prover- and verifier runtime, RainHash2.0 delivers significant improvements. In addition, we prototype RainHash2.0 on FPGA hardware and reach efficiency gains of up to 8.8× over related circuit-friendly hash functions. These results mark RainHash2.0 a practical choice for modern ZK applications.

Keywords: Hash Function · Zero-Knowledge · Binius · VOLEitH · Hardware Acceleration · Binary Extension Fields · RainHash · RainHash2.0

1 Introduction

The area of zero-knowledge proof systems and computational integrity has seen a dynamic and rapid development in the last couple of years. Arithmetization techniques such as *Rank-1 Constraint Systems* (R1CS), *algebraic intermediate representation* (AIR) for

Fast Reed-Solomon Interactive Oracle Proofs of Proximity (FRI)-based commitments [BBHR18], and Plonk-style arithmetizations [GWC19] have made it possible to efficiently verify correctness of computations, potentially even without revealing private data. This evolution has led to many applications, including verifiable computation, privacy-preserving credentials, and blockchain scalability methods such as zkRollups. Most of these proof systems heavily rely on symmetric cryptographic primitives, primarily hash functions, used to construct Merkle tree commitments, instantiate random oracles for Fiat-Shamir heuristics, and compress data for recursive proofs.

Traditional cryptographic hash functions such as the SHA-2 and SHA-3 families are highly optimized for plain evaluation on software and hardware architectures. However, when these functions are represented as arithmetic circuits within a zero-knowledge proving system, they incur a massive computational overhead. For instance, bitwise operations extensively used in SHA-2 are expensive to express in the algebraic constraints required by most proof systems. This bottleneck has led to an ongoing effort in designing *arithmetization-oriented* (AO) or *circuit-friendly* hash functions (e.g., Poseidon [GKR+21] and *Rescue* [AAB+20]) and more specialized proving techniques.

1.1 Binary Fields for Modern Protocols and Efficient Hashing

1.1.1 Binary Field Proof Systems

Modern zero-knowledge proof systems are predominantly defined over prime fields. Large prime fields ($\approx$ 256 bits) are necessary for protocols relying on elliptic curve pairings (e.g., BN254 or BLS12-381, as targeted by the Skyscraper-v2 hash function [BGK+25a, BGK+25b]). Conversely, smaller prime fields (e.g., 31-bit or 64-bit primes) have recently gained popularity for FRI-based zero-knowledge virtual machines (zkVMs) due to their faster base field operations and smaller memory footprint. However, prime field proof systems inherently struggle with binary data and bitwise operations. For example, when emulating a standard CPU architecture (like RISC-V), instructions often operate over 32-bit unsigned integers. To prevent overflow during prime field arithmetic operations, these integers must be split into smaller 8-bit or 16-bit limbs. Embedding these small limbs into 31-bit or 64-bit prime field elements leads to substantial storage waste and a significant arithmetization overhead. To address these fundamental shortcomings, the cryptographic community has seen a rise of interest in *binary* field proof systems.

Recent approaches, most notably the multivariate FRI-Binius framework [DP25, DP24], introduce techniques that decouple the committing field size from the constraint field size. Operating over binary extension towers, Binius enables efficient native bitwise operations, eliminating the embedding overhead.[1] A distinctive feature of Binius is its *ring-switching* mechanism, which allows constraints to be expressed seamlessly across binary extensions of different sizes. This makes it possible to mix components operating at different algebraic sizes within the same proof system without leaving the FRI verification model. Although classical binary-oriented hash functions are more efficient in Binius than in prime-field-based proof systems, they still incur a comparatively large number of constraints. Thus, the need for a native, highly optimized circuit-friendly hash function tailored for binary extension fields remains.

1.1.2 VOLE-in-the-Head and Post-Quantum Signatures

Alternative proof systems focus on ultra-low memory footprints and fast verification without relying on heavy polynomial commitments, unlike state-of-the-art SNARKs such as Binius. An important example is the VOLE-in-the-Head (VOLEitH) paradigm [BBD+23].

[1] Indeed, small values like 4-bit numbers can be naively embedded into a 4-bit field element, instead of requiring 31-bit or 64-bit fields.

VOLEitH has proven efficient for proving small- to medium-sized circuits in zero-knowledge and has become a core building block for plausibly post-quantum digital signature schemes and blind signatures. For example, the POMFRIT blind signature scheme leverages VOLEitH combined with the MAYO signature trapdoor to achieve practical signature sizes and proving times [BBB+26].

The efficiency metrics in a VOLEitH proof differ significantly from those in SNARKs. In VOLEitH, linear operations (such as addition and multiplication by a public constant) are essentially "free" and incur no communication overhead. In contrast, nonlinear operations (multiplications of secret variables) dictate the communication complexity, requiring data proportional to the number of such operations to be transmitted. Consequently, to minimize the signature size and prover time in VOLEitH-based protocols, the underlying cryptographic functions must exhibit extremely low multiplicative complexity. The RainHash function [BBB+26], inspired by the Rain [DKR+22] one-way function, was proposed specifically to cater to MPC and VOLEitH applications. Unlike SHAKE256, whose many rounds and full-state $\mathbb{F}_2$ multiplications lead to high multiplicative complexity and many VOLE constraints, RainHash achieves excellent circuit efficiency by design for VOLEitH/MPC settings. However, its internal structure, relying on wide-field multiplications and dense linear layers over large extension fields, leads to high area utilization and routing traffic when mapped to hardware platforms such as FPGAs or ASICs. Moreover, it exhibits poor AVX plain performance compared to other ZK-friendly hash functions. As VOLEitH-based protocols move toward real-world deployment, the need for a construction which is both circuit-efficient *and* hardware-friendly becomes crucial.

1.1.3 A Focus on Hardware Acceleration

As zero-knowledge proofs are starting to get deployed on a large scale, particularly in applications like zkRollups that process thousands of transactions per second, software implementations are already reaching their physical limits. The trend is shifting towards accelerating heavy ZK computations using dedicated hardware, such as FPGAs and ASICs. Most existing AO hash functions, including early versions of Poseidon, Anemoi [BBC+23], and `Monolith` [GKL+24], were primarily optimized for software execution. Their internal building blocks – mostly large dense *Maximum Distance Separable* (MDS) matrices over prime fields, wide-field multiplications, and sometimes high-degree power maps – translate poorly to hardware. Indeed, they demand excessive lookup table (LUT) utilization, deep critical paths, and complex routing, all of which inflate area and limit clock frequency. A similar limitation applies to the original RainHash: while its low multiplicative depth is attractive for VOLEitH circuit size, its algebraic structure was not designed with silicon efficiency in mind, resulting in worse throughput numbers on FPGAs. Furthermore, RainHash does not natively exploit the structural advantages of binary proof systems like Binius, such as ring-switching between extension fields of different sizes. Hence, designing a primitive that achieves state-of-the-art circuit efficiency while being inherently tailored for efficient hardware acceleration over binary fields represents a critical and timely research direction.

1.1.4 Related Hash Designs over Binary Fields

The design of circuit-friendly hash functions over binary extension fields has only seen limited attempts. Starkad [GKR+19], introduced alongside the original Poseidon, was one of the earliest designs over $\mathbb{F}_{2^n}$. However, it was withdrawn after specific linear layer vulnerabilities to subspace trail attacks were discovered [KR21, BCD+20, GRS21]. The recently proposed Poseidon2b [GKK+26] acts as a secure, binary-field analogue of Poseidon, mitigating these attacks while optimizing for Binius. Similarly, Vision Mark-32 [AMPŠ24] adapts the Rescue/Vision design strategy for binary fields, although it involves

high-degree power maps that can impact native performance. Anemoi [BBC+23] also provides instances over binary fields with odd extension degrees.

The most directly relevant design is RainHash [BBB+26], which targets VOLEitH applications with low multiplicative complexity but whose large-field multiplications and dense linear layers limit its hardware efficiency. While these functions bridge the gap for binary proof systems, none of them simultaneously target the structural advantages of field-switching in Binius combined with hardware-focused design choices.

1.2 Our Contributions

In this paper, we introduce RainHash2.0, a new cryptographic permutation and hash function designed for circuit efficiency in binary ZK proof systems and plain performance in hardware, addressing the performance limitations of RainHash by introducing faster small-field multiplications in the nonlinear layer and a sparser linear layer. Our contributions are summarized as follows.[2]

A New Hardware-Friendly Design over Binary Fields. By focusing on hardware devices, we follow a recent direction of accelerating cryptographic workloads on dedicated hardware. Unlike traditional AO hashes that rely on large, dense MDS matrices over prime fields, RainHash2.0 is natively defined over binary extension fields using an LS-design structure [GLSV15]. The state is viewed as a matrix of bytes, where the nonlinear layer applies parallel inversions over its columns (i.e., the extension field $\mathbb{F}_{2^{64}}$) and the linear layer operates over the base field $\mathbb{F}_{2^8}$ utilizing a lightweight ShiftRows-like mechanism.

Exploiting Binius Ring-Switching. We leverage techniques introduced by the Binius protocol to "horizontally" split the round function. By using finite fields of different sizes simultaneously (performing nonlinear inversions in the larger field $\mathbb{F}_{2^{64}}$ and linear mixing in the smaller base field $\mathbb{F}_{2^8}$) we achieve both fast algebraic degree growth and strong statistical diffusion with a significantly smaller number of total rounds. In our setting, this approach is arguably a novelty in itself and provides a new angle for future arithmetization-oriented designs.

Comprehensive Security Analysis. We provide a detailed security evaluation of RainHash2.0, revisiting statistical attacks (differential, truncated differential, and linear cryptanalysis) and algebraic attacks (Gröbner basis, interpolation). We argue that our field-mixing approach provides good resistance against all known attack vectors.

VOLEitH and Post-Quantum Integration. We demonstrate that the low multiplicative depth and optimized binary structure of RainHash2.0 also make it a good fit for VOLEitH-based zero-knowledge proofs. We evaluate its performance in the context of post-quantum blind signatures, showing that RainHash2.0 significantly reduces the communication overhead and signature size compared to standard functions like SHAKE256 and improves upon the original RainHash construction in terms of plain performance.

Benchmarks. We discuss native hardware implementations of RainHash2.0 on modern FPGAs, showing up to 8.85× better hardware efficiency compared to existing circuit-friendly hash functions. At the same time, our hardware implementation consumes up to 3.8× less area and maintains a high clock frequency of 500 MHz to guarantee performant hashing throughput. We also provide proof implementations within the Binius framework. We show the plain optimized performance of RainHash2.0 is at-par with industry standard

[2] All implementations and experimental results are available at https://extgit.isec.tugraz.at/krypto/rainhash2.

Poseidon hash function and is 8× times faster than RainHash for hashing similar sized message. RainHash2.0 also out-performs SHAKE by ≈ 2× times in terms of VOLEitH proof size and prove and verify runtime ≈ 44× and ≈ 11× respectively. Compared to RainHash prove/verify runtime, RainHash2.0 is ≈ 6× times faster.

1.3 Organization and Notation

Section 2 recalls binary tower fields, Binius, and the VOLEitH framework. Section 3 specifies the RainHash2.0 permutation and supported modes of operation; Section 4 gives a detailed overview of our design rationale, including a comparison to related designs. Section 5 provides a security analysis covering both state-of-the-art statistical and algebraic attack vectors. Section 6 outlines the proof-system realizations in both Binius and VOLE circuits, while Section 7 concludes the paper with a description of the hardware implementations. Concrete benchmarks are provided in the corresponding sections.

We use the following conventions throughout; section-specific notation is introduced where needed. Blackboard-bold letters denote finite fields, with $\mathbb{F}$ used for general finite fields and $\mathbb{T}$ for tower fields. We write $\mathbb{F}_{2^n}$ for the finite field with 2^n elements, and $\mathbb{T}_k$ for the binary tower field isomorphic to $\mathbb{F}_{2^{2^k}}$. For a field extension of degree m, we identify $\mathbb{F}_{2^{nm}}$ with the vector space $\mathbb{F}_{2^n}^m$ via some fixed basis. Boldface letters denote coordinate vectors over finite fields, e.g., $\mathbf{x} = (x_0, \ldots, x_{m-1}) \in \mathbb{F}_{2^n}^m$. We use sans-serif letters for functions (e.g., F, with P for permutations) and calligraphic letters for sets. Throughout, R is the number of rounds, t the state size, and indices run $0 \le r < \mathrm{R}$, $0 \le j < t$.

2 Background

In this section, we first recall binary tower fields as a concrete and implementation-friendly realization of binary extension fields. We then summarize the Binius proof system and the VOLEitH-based NIZK framework, which are the two primary application targets of our design. Finally, we review the original RainHash construction which serves as a basis for our new RainHash2.0 design.

2.1 Binary Tower Fields

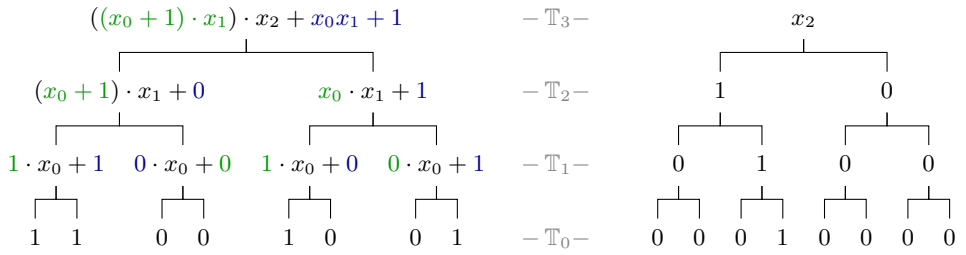
In this section, we briefly recall the Wiedemann tower [Wie88], a binary tower field construction that has recently been adopted by designs targeting the Binius proof system [AMPŠ24, GKK+26]. Let $\mathbb{T}_0 := \mathbb{F}_2$. For $k > 0$, the field

$$\mathbb{T}_k = \mathbb{T}_{k-1}[x_{k-1}]/\mathsf{F}_k(x_{k-1})$$

is defined as a quadratic extension of $\mathbb{T}_{k-1}$, where $\mathsf{F}_k(x_{k-1}) = x_{k-1}^2 + x_{k-2}x_{k-1} + 1$ is irreducible over $\mathbb{T}_{k-1}[x_{k-1}]$. Consequently, for all $k \ge 0$, the field $\mathbb{T}_k$ is isomorphic to $\mathbb{F}_{2^{2^k}}$. Moreover, for integers $0 \le k_1 < k_2$, the field $\mathbb{T}_{k_2}$ can be viewed as a degree-m extension of $\mathbb{T}_{k_1}$ with $m = 2^{k_2-k_1}$. Hence, conversion between the vector space $\mathbb{T}_{k_1}^m$ and the field $\mathbb{T}_{k_2}$ is straightforward, where a natural basis is given by the monomials in the tower generators with exponents in $\{0, 1\}$.

For $k \ge 1$, any $a \in \mathbb{T}_k$ can be written as $a = a_1 \cdot x_{k-1} + a_0$ with $a_0, a_1 \in \mathbb{T}_{k-1}$. This representation reflects the recursive structure of the tower, illustrated in Figure 1, which, as we will see, carries over to arithmetic operations and enables more efficient implementations of multiplication and inversion.

Binary tower field arithmetic [FP97]. Let $a = a_1 \cdot x_{k-1} + a_0$ and $b = b_1 \cdot x_{k-1} + b_0$ be elements of $\mathbb{T}_k$ for $k \ge 1$, where $a_0, a_1, b_0, b_1 \in \mathbb{T}_{k-1}$. In the following, we express the complexities of arithmetic in $\mathbb{T}_k$ recursively in terms of operations in $\mathbb{T}_{k-1}$ and over $\mathbb{F}_2$.



(a) Element $((x_0+1)\cdot x_1)\cdot x_2 + x_0x_1 + 1 \in \mathbb{T}_3$. **(b)** Constant x_2 in $\mathbb{T}_3$.

Figure 1: Recursive decomposition of elements in $\mathbb{T}_3 \cong \mathbb{F}_{2^8}$ into subfield elements.

- *Addition* A_k. One addition in $\mathbb{T}_k$ requires two additions in $\mathbb{T}_{k-1}$, since

$$a + b = (a_1 + b_1)\cdot x_{k-1} + (a_0 + b_0)\,. \tag{1}$$

  Recursively, we get $\mathrm{A}_k = 2^k\mathrm{A}_0$, where A_0 denotes the complexity of adding two $\mathbb{F}_2$ elements. Thus, addition in $\mathbb{T}_k$ is simply realized as the bitwise `XOR` of the corresponding binary vectors.

- *Constant multiplication* C_k. Multiplication by the generator x_{k-1} satisfies

$$a\cdot x_{k-1} = (a_1\cdot x_{k-1} + a_0)\cdot x_{k-1} = (a_1\cdot x_{k-2} + a_0)\cdot x_{k-1} + a_1\,, \tag{2}$$

  using $x_{k-1}^2 = x_{k-2}x_{k-1} + 1$. Thus, $\mathrm{C}_k = \mathrm{C}_{k-1} + \mathrm{A}_{k-1} = \mathrm{C}_0 + (2^k - 1)\mathrm{A}_0$, where C_0 denotes the complexity of multiplication by one in $\mathbb{F}_2$.

- *Multiplication* M_k. Using a Karatsuba-like approach, multiplication can be realized via three multiplications, one constant multiplication, and some additions in $\mathbb{T}_{k-1}$:

$$a\cdot b = (a_0b_1 + a_1b_0 + a_1b_1\cdot x_{k-2})\cdot x_{k-1} + a_0b_0 + a_1b_1\,. \tag{3}$$

  Hence, $\mathrm{M}_k = 3\mathrm{M}_{k-1} + 6\mathrm{A}_{k-1} + \mathrm{C}_{k-1} = 3^k\mathrm{M}_0 + 6(3^k - 2^k)\mathrm{A}_0 + \frac{3^k-1}{2}(\mathrm{C}_0 - \mathrm{A}_0)$, where M_0 corresponds to the cost of multiplying two $\mathbb{F}_2$ elements, i.e., one AND operation.

- *Squaring* S_k. Squaring in $\mathbb{T}_k$ is notably cheaper than simple multiplication since

$$a^2 = (a_1^2\cdot x_{k-2})\cdot x_{k-1} + a_0^2 + a_1^2\,, \tag{4}$$

  yielding $\mathrm{S}_k = 2\mathrm{S}_{k-1} + \mathrm{C}_{k-1} + \mathrm{A}_{k-1} = 2^k\mathrm{S}_0 + k2^k\mathrm{A}_0 + (2^k - 1)(\mathrm{C}_0 - \mathrm{A}_0)$, where $\mathrm{S}_0 = 0$ since squaring in $\mathbb{F}_2$ is trivial.

- *Inversion* I_k. The inverse of $a \in \mathbb{T}_k$ is given by

$$a^{-1} = (a_1\Delta^{-1})\cdot x_{k-1} + (a_0 + a_1\cdot x_{k-2})\cdot\Delta^{-1}\,, \tag{5}$$

  where $\Delta = a_0(a_0 + a_1\cdot x_{k-2}) + a_1^2 \in \mathbb{T}_{k-1}$. Thus, $\mathrm{I}_k = \mathrm{I}_{k-1} + 3\mathrm{M}_{k-1} + 2\mathrm{A}_{k-1} + \mathrm{C}_{k-1} + \mathrm{S}_{k-1}$, where I_0 is negligible since inversion in $\mathbb{F}_2$ is trivial. Note that the complexity of a single inversion is asymptotically about 1.5 times that of a multiplication:

$$\mathrm{I}_k \approx \mathrm{I}_{k-1} + 3\mathrm{M}_{k-1} \approx \sum_{i=0}^{k-1} 3\mathrm{M}_i \approx 3\mathrm{M}_0\cdot\sum_{i=0}^{k-1} 3^i = \frac{3}{2}(3^k - 1)\mathrm{M}_0 \approx \frac{3}{2}\cdot\mathrm{M}_k\,,$$

  where we ignore lower-order terms and use $\mathrm{M}_k \approx 3\mathrm{M}_{k-1} \approx 3^k\mathrm{M}_0$.

2.2 Binius

Binius is a state-of-the-art SNARK that extends Plonkish polynomial commitment frameworks using FRI-based polynomial interactive oracle proofs (IOPs). In Binius, computational traces are represented as polynomials derived from an execution trace table, which is both constrained and committed within the protocol. The prover provides oracle access to these committed polynomials, enabling the verifier to issue queries during the proof verification process.

A defining characteristic of Binius is its use of binary tower fields, enabling computations over fields isomorphic to $\mathbb{F}_{2^{2^k}}$ for $k \geq 0$. In contrast to traditional constructions over large prime fields, where the field size is fixed by a chosen prime, this approach allows flexible field sizes and avoids embedding overhead, which is particularly beneficial for small fields. Furthermore, the ring-switching protocol enables dynamic transitions between field sizes within a single constraint system via a packing operation that maps elements from smaller binary extension fields into a 128-bit ring representation, enabling efficient computation across different extension fields.

2.3 VOLEitH based NIZK

A zero-knowledge (ZK) proof of knowledge [GMR85] allows a prover P to convince a verifier V of the validity of a statement, using a public input (public data and the circuit) and potentially private witness data such that the public input reveals no information about the statement.[3] A ZKP must satisfy three key properties: *completeness*, *soundness*, and *zero knowledge*. Informally, completeness ensures that an honest P can convince an honest V, while soundness guarantees that any dishonest P can only convince an honest V with negligible probability. The zero-knowledge property ensures that the V learns nothing about the witness. ZK proofs can be made non-interactive using transformations such as Fiat-Shamir [FS87] and Fischlin [Fis05]. The former is widely used in digital signature constructions, including post-quantum schemes based on MPC-in-the-Head (MPCitH) [CDG+17, DKR+22] and VOLE-in-the-Head (VOLEitH) [BBD+23, BBM+24] techniques.

VOLEitH [BBD+23], along with its later optimized versions [BBM+24, BBB+25d], is a non-interactive zero-knowledge (NIZK) proof system with applications in efficient post-quantum signatures and in proving small- to medium-sized circuits in ZK [BOS+26]. It is based on *vector oblivious linear evaluation* (VOLE) correlations: a VOLE instance of length m is a two-party correlation between a prover P and a verifier V, where P holds random bits $\mathbf{u} \in \mathbb{F}_2^m$ and a VOLE tag $\mathbf{v} \in \mathbb{F}_{2^k}^m$, while V holds a global key $\Delta \in \mathbb{F}_{2^k}$ and a VOLE key $\mathbf{q} \in \mathbb{F}_{2^k}^m$ such that $\mathbf{q} = \mathbf{u} \cdot \Delta + \mathbf{v}$, with $k = \lambda$ determined by the desired security level. Since V must keep Δ hidden (otherwise P could adapt $\mathbf{v}$, breaking soundness), VOLE-based protocols are inherently *designated verifier* [JSI96] and do not directly yield publicly verifiable proofs. VOLEitH [BBD+23] overcomes this limitation using a *delayed* strategy, deriving Δ via the Fiat-Shamir transform together with all-but-one vector commitments, independently of P's inputs, which enables applications such as digital signatures. The VOLE correlation tuple $(\mathbf{u}, \mathbf{v}, \mathbf{q})$ can be viewed as a linearly homomorphic commitment to $\mathbf{u}$, enabling efficient VOLE-based ZK proof systems [BDSW23].

Using the *Quicksilver* proof strategy [YSWW21] in the VOLEitH protocol, linear gates (addition with secret or constant and multiplication with constant) incur no communication overhead, while nonlinear gates (multiplications with secret values) require communication linear in their number and thus dominate the proof size. Consequently, reducing the proof size requires minimizing the multiplicative complexity (count and depth) of the underlying circuit, motivating the design of *MPC-/ZK-/arithmetization-friendly*

[3] For instance, P may want to convince V that given a cryptographic hash function H and a publicly known value y they know a preimage x such that $y = \mathsf{H}(x)$.

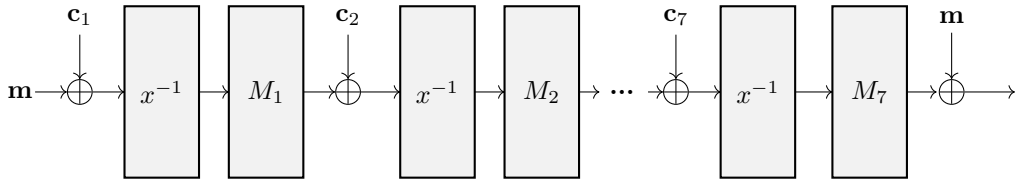


Figure 2: RainHash [BBB+26] construction with 7 rounds of the Rain permutation. The input $\mathbf{m}$ is XOR-ed after the final round. In round i, $\mathbf{c}_i$ denotes the round constant and M_i denotes the multiplication with an unstructured invertible matrix over $\mathbb{F}_2$.

circuits. Such circuits often represent cryptographic primitives like hash functions used in applications like membership-proofs, verifiable-computation, anonymous credentials, among others. Arithmetization-friendly hash functions have been designed for various applications like MPC [ARS+15], zkSNARKs [GKR+21, BGK+25a], and more recently for VOLE-ZK [BBB+26].[4]

2.4 RainHash

RainHash [BBB+26] is an MPC-/VOLEitH-friendly hash function based on the one-way-function (OWF) Rain [DKR+22]. The original designers of the OWF proposed it for constructing the MPCitH-based post-quantum (PQ) signature Rainier and it was later adopted in the VOLEitH-based PQ signature MandaRain [BBM+24]. For the blind signature use-case, the authors show that adopting RainHash for 128-bit security, PoMFRIT blind signatures, achieve both reduced signature size and prover/verifier runtime compared to using SHAKE.

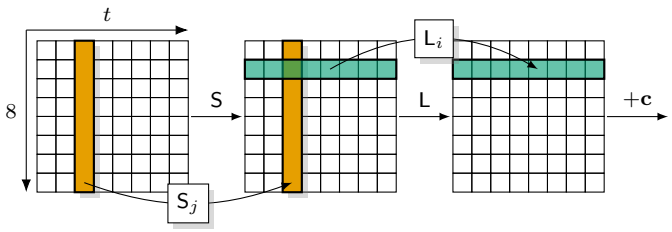
RainHash Construction. RainHash uses essentially the same round function as the Rain OWF, with 7 rounds and no special treatment of the final round. In contrast to Rain, where the last linear layer is omitted (since an attacker could invert it via the known matrix inverse), RainHash retains this layer to improve diffusion. This is possible because the output is no longer trivially invertible: the input message is XOR-ed to the state after the final linear layer, a common paradigm in short-input hashing [KLMR16] inspired by the Davies-Meyer [Mat85] construction replacing the keyed permutation with a "fixed key" one. We refer to Figure 2 for the construction.

RainHash Limitations. Polynomial multiplication and reduction have quadratic complexity, making the large-field inversion S-box in RainHash a major bottleneck for plain performance, especially compared to other ZK-friendly hash functions (cf. Table 9). Additionally, while increasing the state size would allow RainHash to absorb more message bits, this requires S-boxes over even larger fields, worsening this issue. Moreover, large-field inversions are problematic for hardware implementations due to high area utilization and routing overhead (cf. Table 11). In RainHash2.0, we thus replace large-field S-boxes with smaller-field S-boxes, resulting in a significant speed-up in plain evaluation.

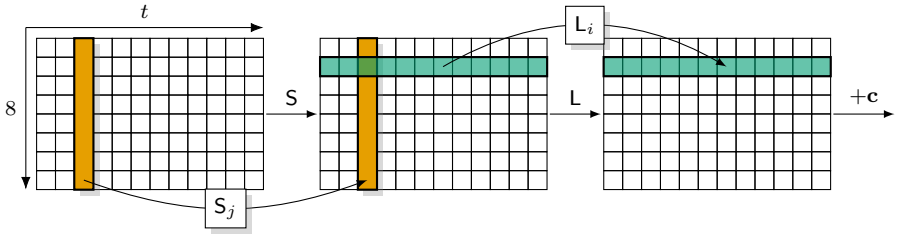
3 RainHash2.0

This section provides an overview of the RainHash2.0 permutation and the accompanying modes of operation. In the following, we write $\mathbb{F}_{2^n}$ to denote a finite field with 2^n

[4] Specialized hash designs are not uncommon and have been explored for hardware-, software-, and zkSNARKs-friendly use cases, e.g., [ANWW13, DEMS21, GKR+21, GKS23, GKL+24, GKK+26].



(a) One round of $\mathsf{P}_{64,t}$ with $t = 8$ for a total state size of 512 bits.



(b) One round of $\mathsf{P}_{64,t}$ with $t = 12$ for a total state size of 768 bits.

Figure 3: One round of the RainHash2.0 permutation over $\mathbb{F}_{2^{64}}^t$. The internal state is represented as a matrix of 8-bit field elements, where each column groups eight such elements and can be interpreted as a 64-bit field element.

elements, agnostic of the concrete field representation. For efficiency reasons, concrete implementations use binary tower fields, which enable efficient decomposition of field elements, offer hardware-friendly arithmetic, and are compatible with the Binius proof system.

3.1 RainHash2.0 Permutation

The RainHash2.0 family includes two classes of permutations, namely $\mathsf{P}_{64,t}$ defined over $\mathbb{F}_{2^{64}}^t \cong \mathbb{F}_{2^8}^{8\times t}$ for $t \in \{8, 12\}$, allowing a total state size of either 512 or 768 bits. Its design principle is similar to the LS design [GLSV15]. The state is interpreted as a rectangle of $8 \times t$ elements in $\mathbb{F}_{2^8}$, that is, 8 rows and t columns, where each cell is a byte-element. Each column can equivalently be viewed as an element of the extension field $\mathbb{F}_{2^{64}}$ via the natural isomorphism $\Phi : \mathbb{F}_{2^8}^8 \to \mathbb{F}_{2^{64}}$ arising from the field extension $\mathbb{F}_{2^{64}}/\mathbb{F}_{2^8}$ of degree 8. Then, in every round, a nonlinear function (or S-box) S operates over the columns, while a linear function L operates over the rows. A graphical representation of the RainHash2.0 permutation is given in Figure 3.

3.1.1 Details of the Round Function

The permutation $\mathsf{P}_{64,t} : \mathbb{F}_{2^{64}}^t \to \mathbb{F}_{2^{64}}^t$ consists of R rounds $\mathsf{R}^{(r)}$, with an additional linear operation L applied to the input, that is,

$$\mathsf{P}_{64,t} = \mathsf{R}^{(\mathrm{R}-1)} \circ \mathsf{R}^{(\mathrm{R}-2)} \circ \ldots \circ \mathsf{R}^{(0)} \circ \mathsf{L} \quad \text{with} \quad \mathsf{R}^{(r)} = \mathbf{c}_r + \mathsf{L} \circ \mathsf{S} \tag{6}$$

for all $0 \le r < \mathrm{R}$, where $\mathbf{c}_r \in \mathbb{F}_{2^{64}}^t$ are the round constants and S, L are defined as follows:

- The nonlinear layer $\mathsf{S} : \mathbb{F}_{2^{64}}^t \to \mathbb{F}_{2^{64}}^t$ operates over the state-vector in $\mathbb{F}_{2^{64}}$, that is, the columns of the $8 \times t$ state-matrix. The coordinate functions S_j are given by

$$\mathsf{S}_j : \mathbb{F}_{2^{64}} \to \mathbb{F}_{2^{64}} \quad \text{with} \quad x \mapsto x^{-1} \tag{7}$$

  for all $0 \le j < t$ and $0^{-1} := 0$ as usual.

- The linear layer $\mathsf{L} : \mathbb{F}_{2^8}^{8\times t} \to \mathbb{F}_{2^8}^{8\times t}$ operates directly over the rows of the $8 \times t$ state-matrix. The coordinate functions L_i are given by

$$\mathsf{L}_i : \mathbb{F}_{2^8}^t \to \mathbb{F}_{2^8}^t \quad \text{with} \quad \mathbf{x} \mapsto (\mathbf{x} \ggg i) \oplus \alpha \cdot (\mathbf{x} \lll i) \tag{8}$$

  for all $0 \le i < 8$, where $\alpha \in \mathbb{F}_{2^8}$ is chosen such that L is invertible (cf. Proposition 1), and $\cdot \ggg i$ and $\cdot \lll i$ denote the right and left rotation by a factor i, that is,

$$(x_0, x_1, \ldots, x_{t-1}) \ggg i \coloneqq (x_{t-i}, x_{t-i+1}, \ldots, x_{t-1}, x_0, \ldots, x_{t-i-1}),$$
$$(x_0, x_1, \ldots, x_{t-1}) \lll i \coloneqq (x_i, x_{i+1}, \ldots, x_{t-1}, x_0, \ldots, x_{i-1}).$$

  Depending on the state size t, the parameter α is defined as follows:

  - $t = 8$: $\alpha = 0$, that is, the linear layer is defined analogously to the ShiftRows operation in AES [DR02a];
  - $t = 12$: $\alpha \neq 0$; the concrete choice depends on the chosen field construction (see Remark 2 in Section 4.3).

Remark 1. Before proceeding, we remark that in the above design description we implicitly switch between the representations over $\mathbb{F}_{2^{64}}^t$ and $\mathbb{F}_{2^8}^{8\times t}$ via the isomorphisms $\Phi : \mathbb{F}_{2^8}^8 \to \mathbb{F}_{2^{64}}$ and Φ^{-1}. That is, we freely interpret columns of the state matrix as elements of the extension field and vice versa, without making these conversions explicit, in order to improve readability.

3.1.2 Round Constant Generation

The round constants are generated deterministically from the fractional part of π. More precisely, we extract a sequence of $t\,\mathrm{R}$ elements in $\mathbb{F}_{2^{64}}$ by taking consecutive blocks of 64 bits from the binary expansion of $\pi - 3$. These bits are interpreted as elements of $\mathbb{F}_2^{64}$ and mapped to $\mathbb{F}_{2^{64}}$ via the chosen basis representation. The resulting constants are arranged into R vectors of length t, yielding $\mathbf{c}_r \in \mathbb{F}_{2^{64}}^t$ for $0 \le r < \mathrm{R}$.

3.2 Modes of Operation

Let χ denote the digest length and t the state size, both measured in elements of $\mathbb{F}_{2^{64}}$. We propose two modes of operation:

- *Sponge*: a hash function for variable-length inputs, $\mathsf{H} : \mathbb{F}_{2^{64}}^* \to \mathbb{F}_{2^{64}}^\chi$.
- *Compression*: a fixed-size compression function $\mathsf{C} : \mathbb{F}_{2^{64}}^t \to \mathbb{F}_{2^{64}}^\chi$.

For fixed-size inputs, such as in Merkle tree constructions, the sponge is typically instantiated with a single permutation call. In this setting, a compression function is more efficient, as it eliminates the need for a capacity part and allows for smaller state sizes. In the following, we describe how these constructions are instantiated from a permutation $\mathsf{P} : \mathbb{F}^t \to \mathbb{F}^t$ with a target security level of κ bits, where $n = \lceil \log_2 |\mathbb{F}| \rceil$ denotes the field size in bits. The proposed parameter sets are summarized in Table 1.

Notation. We write $\mathbf{x} \,\|\, \mathbf{y} \in \mathbb{F}^{\ell_x + \ell_y}$ for the concatenation of two tuples $\mathbf{x} \in \mathbb{F}^{\ell_x}$ and $\mathbf{y} \in \mathbb{F}^{\ell_y}$, and denote by $\mathsf{left}_\ell(\mathbf{x})$ the length-ℓ prefix of $\mathbf{x}$. In particular, $\mathsf{left}_{\ell_x}(\mathbf{x} \,\|\, \mathbf{y}) = \mathbf{x}$.

Table 1: Possible parameters for different state sizes t and a security level of $\kappa = 128$ bits. Input and output sizes are measured in elements of $\mathbb{F}_{2^{64}}$.

		Sponge			Compression		
κ	state size t	capacity c	rate r	digest size χ	$t-\chi$	χ	Arity
128	8 (512 bit)	4 (256 bit)	4 (256 bit)	≥ 4 (256 bit)	4 (256 bit)	4 (256 bit)	2:1
128	12 (768 bit)	4 (256 bit)	8 (512 bit)	≥ 4 (256 bit)	8 (512 bit)	4 (256 bit)	3:1

3.2.1 Sponge

The sponge construction [BDPV08] is a widely used paradigm for hashing variable-length inputs. We write $t = r + c$, where c and r denote the *capacity* and *rate*, respectively, both measured in field elements. The sponge construction proceeds in two phases.

- In the *absorption phase*, a (padded) message $\mathbf{M} \in \mathbb{F}^*$ is split into blocks $\mathbf{M}_i \in \mathbb{F}^r$ for $0 \leq i < \ell$, which are iteratively absorbed into the state $\mathbf{S} \in \mathbb{F}^t$. Starting from the initialization $\mathbf{S} \leftarrow \mathbf{0}^{c-1} \,\|\, \chi$, each block is processed as

$$\mathbf{S} \leftarrow \mathsf{P}\big(\mathbf{S} + (\mathbf{M}_i \,\|\, \mathbf{0}^c)\big)\,. \tag{9}$$

- In the *squeezing phase*, the output is generated by repeatedly applying P and extracting the first r elements of the state until χ output elements are obtained.

We use the sponge-pi construction [LBM25], illustrated in Figure 4, where the final absorption step is modified to include a padding indicator $\mu \in \{0, 1\}$. Concretely, the last block is absorbed as

$$\mathbf{S} \leftarrow \mathsf{P}\big(\mathbf{S} + (\mathbf{M}_{\ell-1} \,\|\, \mathbf{0}^{c-1} \,\|\, \mu)\big)\,, \tag{10}$$

ensuring domain separation between padded and unpadded inputs.

Security Requirements. Assuming the underlying permutation P behaves like a random permutation, the sponge construction is indifferentiable from a random oracle up to about $2^{nc/2}$ queries [BDPV08]. Thus, to achieve a (pre-quantum) security level of κ bits, a sponge hash function $\mathsf{H} : \mathbb{F}^* \to \mathbb{F}^\chi$ with rate r, capacity c, and digest length χ must satisfy $c \geq \lceil \frac{2\kappa}{n} \rceil$, and $\chi \geq \lceil \frac{2\kappa}{n} \rceil$ to prevent collision attacks. Finally, the security of a sponge hash function is related to the infeasibility of solving any CICO (*Constrained Input Constrained Output*) problem for the permutation that instantiates the sponge:

Definition 1 (CICO problem [BDPA11a])**.** Let $\mathsf{P} : \mathbb{F}^t \to \mathbb{F}^t$ be a permutation. Given possible input and output sets $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{F}^t$, the CICO problem asks to find (x, y) with $y = \mathsf{P}(x)$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$.

In practice, the constraints fix k_{in} input coordinates and k_{out} output coordinates (where $0 < k_{\text{in}}, k_{\text{out}} < t$ and $k_{\text{in}} + k_{\text{out}} \leq t$) to prescribed values, giving $|\mathcal{X}| = |\mathbb{F}|^{t-k_{\text{in}}}$ and $|\mathcal{Y}| = |\mathbb{F}|^{t-k_{\text{out}}}$. We refer to this as the CICO-$(k_{\text{in}}, k_{\text{out}})$ problem, and write CICO-k for the symmetric case $k_{\text{in}} = k_{\text{out}} = k$. The generic brute-force cost is $\min(|\mathbb{F}|^{k_{\text{in}}}, |\mathbb{F}|^{k_{\text{out}}})$ evaluations of P or P^{-1}.

3.2.2 Compression

In the compression setting, we use a permutation-based variant of the Davies-Meyer construction, as already employed for Poseidon2 [GKS23] and Poseidon(2)b [GKK+26],

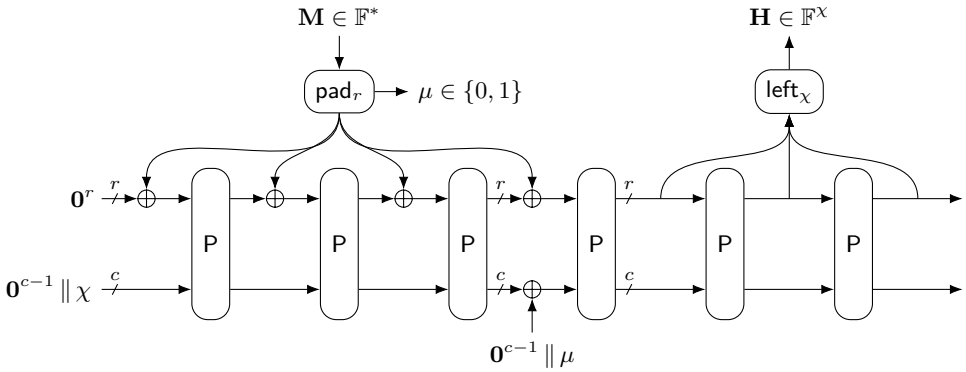


Figure 4: sponge-pi [LBM25] using a permutation $\mathsf{P} : \mathbb{F}^t \to \mathbb{F}^t$ with $t = r + c$.

defined by combining the truncation function with the feed-forward operation. The permutation P is applied once, followed by feed-forward addition and truncation:

$$\mathsf{C} : \mathbb{F}_{2^{64}}^t \to \mathbb{F}_{2^{64}}^\chi , \quad \mathbf{x} \mapsto \mathsf{left}_\chi\big(\mathsf{P}(\mathbf{x}) + \mathbf{x}\big). \tag{11}$$

Viewing the input $\mathbf{x} \in \mathbb{F}^t$ as the concatenation of a message part $\mathbf{x}_m \in \mathbb{F}^{t-\chi}$ and a chaining value $\mathbf{x}_c \in \mathbb{F}^\chi$, we get a compression function of arity $t : \chi$ of the form

$$\mathsf{C}_{t:\chi} : \mathbb{F}^{t-\chi} \times \mathbb{F}^\chi \to \mathbb{F}^\chi , \quad (\mathbf{x}_m, \mathbf{x}_c) \mapsto \mathsf{C}(\mathbf{x}_m \,\|\, \mathbf{x}_c) . \tag{12}$$

In particular, for $\chi = t/2$, this corresponds to a 2-to-1 compression function, while for $\chi = t/3$, it yields a 3-to-1 compression function.

Security Requirements. To achieve a (pre-quantum) security level of κ bits, a compression function $\mathsf{C} : \mathbb{F}^t \to \mathbb{F}^\chi$ must satisfy $\chi \geq \left\lceil \frac{2\kappa}{n} \right\rceil$ to prevent collision attacks and $t - \chi \geq \left\lceil \frac{\kappa}{n} \right\rceil$ to avoid guessing attacks on the truncated part. As the entire permutation input is hashed, domain separation is not provided by the construction and must be implemented externally, if needed.

3.3 Number of Rounds and Security Claims

We claim that the hash/compression function instantiated by 8-round RainHash2.0 provides 128 bits of security with respect to (second) pre-image and collision attacks. Moreover, to ensure the security of the sponge construction (with rate r and capacity c), we claim that the underlying permutation is CICO-k secure for all $1 \leq k \leq c$, i.e., that no attack solves the CICO-k problem (cf. Definition 1) faster than $\min\{2^{64k}, 2^{128}\}$.

Note that the number of rounds for the RainHash2.0 permutation is equal to 8 for both $t = 8$ and $t = 12$ (independently of the mode of operation). A detailed security analysis is given in Section 5.

4 Design Rationale

In this section, we explain the rationale behind our construction and discuss the main design choices.

4.1 Functions over Different Fields

The main idea behind our design strategy is to combine functions over different fields to achieve security within only a few rounds. In this section, we review prior designs following

this approach and analyze their respective advantages and disadvantages. These insights informed the design of RainHash2.0 as a weak-arranged *Substitution-Permutation-Network* (SPN) scheme with an S-box S over $\mathbb{F}_{2^{64}}$ and a linear layer L operating at byte level.

Weak-Arranged SPN Schemes. One approach is to define the nonlinear and linear layers over different fields. This strategy is not new in the literature. For example, the S-box of PRESENT [BKL+07] is defined over $\mathbb{F}_{2^4}$, whereas its linear layer is a bit permutation and therefore has degree one over $\mathbb{F}_2$. Similarly, the S-box of Keccak/SHA-3 (i.e., the χ function) acts over $\mathbb{F}_{2^5}$, while its linear layer has degree one over $\mathbb{F}_2$. Following [CGG+22, Definition 2], both primitives are examples of so-called *weak-arranged SPN schemes*.

Weak-arranged SPN schemes offer some clear advantages in the context of algebraic attacks. Consider an SPN defined over $\mathbb{F}_{2^n}^t$ whose S-box is the power map $x \mapsto x^d$ over $\mathbb{F}_{2^n}$ and the linear layer is defined by a matrix multiplication over $\mathbb{F}_2$. As shown in [BCD11, CGG+22], its degree grows substantially faster than in a *strong-arranged SPN scheme* where, roughly speaking, the linear layer is also defined over $\mathbb{F}_{2^n}$.

Non-Linear Maps over Different Fields (XHash). The idea of field-mixing can be taken further by combining nonlinear operations defined over different fields, for example, some over $\mathbb{F}_{2^n}$ and others over $\mathbb{F}_{2^{nm}}$. A concrete example of this approach is XHash [ABK+23, ABK+26], designed by Ashur et al. for STARK applications: it applies $x \mapsto x^d$ (as well as $x \mapsto x^{1/d}$) over $\mathbb{F}_{2^n}$ and $X \mapsto X^d$ over $\mathbb{F}_{2^{3n}}$. In the following discussion, we adopt their notation to clearly distinguish the power maps over the two fields, using lowercase variables for elements of $\mathbb{F}_{2^n}$ and uppercase variables for elements of $\mathbb{F}_{2^{nm}}$.

On the positive side, $X \mapsto X^d$ has a dense polynomial representation over $\mathbb{F}_{2^n}^m$, which is crucial for guaranteeing security against algebraic attacks (see, e.g., [ABK+26, App. D]). Moreover, this choice is advantageous for achieving rapid degree growth over $\mathbb{F}_2$. As shown in [EGL+20, CGG+22], for a symmetric primitive over $\mathbb{F}_{2^n}^t$ with a nonlinear layer $x \mapsto x^d$ defined over $\mathbb{F}_{2^n}$ and a linear layer defined over $\mathbb{F}_{2^n}^t$, the degree grows linearly with the number of rounds. By combining power maps defined over $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^{nm}}$, the degree over $\mathbb{F}_2$ grows more rapidly.

However, this approach also has several disadvantages. First, the degree of the polynomial representation of $X \mapsto X^d$ over $\mathbb{F}_{2^n}^m$ is again d. Hence the growth of the degree of an R-round SPN over $\mathbb{F}_{2^n}$ is again d^R, yielding no asymptotic advantage in degree growth. Yet $X \mapsto X^d$ over $\mathbb{F}_{2^{nm}}$ is substantially more expensive than m power maps $x \mapsto x^d$ over $\mathbb{F}_{2^n}$, degrading plain performance. Finally, ensuring invertibility of $X \mapsto X^d$ imposes additional constraints on both n and m, with a potential impact on the design choices.

4.2 About the Nonlinear Layer: Monomial Power Maps

Several options are possible for the nonlinear layer. To minimize the *multiplicative complexity*, one of our primary design goals, we use an invertible power map over, we opted for an invertible power map over $\mathbb{F}_{2^n}$. In the following, we briefly review the main options considered in the literature and explain the considerations that led us to choose the inverse map for RainHash2.0.

Interleaving x^d and $x^{1/d}$ (*Rescue*-like). Several designs, including *Rescue* [AAB+20], Griffin [GHR+23], and Anemoi [BBC+23], use both $x \mapsto x^d$ and $x \mapsto x^{1/d}$, where $d \geq 3$ is typically chosen as the smallest integer such that $x \mapsto x^d$ is invertible. Since these schemes are strong-arranged, using only $x \mapsto x^d$ would lead to comparatively slow degree growth and therefore require many rounds to reach maximal degree. The additional use of $x \mapsto x^{1/d}$ accelerates the degree growth, allowing a high algebraic degree to be reached within only a few rounds. However, evaluating $x \mapsto x^{1/d}$ is costly in both software and

Table 2: Theoretic costs of computing x^7 vs. x^{-1} in $\mathbb{T}_6$. See Section 2.1 for notation.

Operation	$\mathbb{T}_6 \cong \mathbb{F}_{2^{64}}$	
	$\mathbb{T}_0 \cong \mathbb{F}_2$	$\mathbb{T}_3 \cong \mathbb{F}_{2^8}$
$x \mapsto x^7$	$7894\mathrm{A}_0 + 854\mathrm{C}_0 + 1458\mathrm{M}_0 + 128\mathrm{S}_0$	$236\mathrm{A}_3 + 40\mathrm{C}_3 + 54\mathrm{M}_3 + 16\mathrm{S}_3$
$x \mapsto x^{-1}$	$5265\mathrm{A}_0 + 600\mathrm{C}_0 + 1092\mathrm{M}_0 + 63\mathrm{S}_0 + \mathrm{I}_0$	$117\mathrm{A}_3 + 22\mathrm{C}_3 + 39\mathrm{M}_3 + 7\mathrm{S}_3 + \mathrm{I}_3$

Table 3: Hardware costs of computing x^7 (implemented using 2 multiplications and 2 squarings) vs. x^{-1} over different tower field sizes. Synthesis results of the non-pipelined modules are from Vivado 2022.2 with area optimizations enabled.

Field Size	Bits per Element	Area Consumption (LUTs)				Benefit of
		Multiply	Square	$x \mapsto x^7$	$x \mapsto x^{-1}$	$x \mapsto x^{-1}$
$\mathbb{T}_3 \cong \mathbb{F}_{2^8}$	8	30	4	32	27	1.19×
$\mathbb{T}_4 \cong \mathbb{F}_{2^{16}}$	16	101	11	223	116	1.92×
$\mathbb{T}_5 \cong \mathbb{F}_{2^{32}}$	32	342	40	730	416	1.75×
$\mathbb{T}_6 \cong \mathbb{F}_{2^{64}}$	64	1,097	100	2,390	1,399	1.71×
$\mathbb{T}_7 \cong \mathbb{F}_{2^{128}}$	128	3,432	238	7,437	4,461	1.67×

hardware. Using a weak-arranged scheme instead enables rapid degree growth, making the additional cost of $x \mapsto x^{1/d}$ unnecessary. We therefore do not consider a *Rescue*-like nonlinear layer. In particular, we expect RainHash2.0 to achieve maximum degree within few rounds due to the particular choice of the linear layer, while avoiding the high cost of $x \mapsto x^{1/d}$.[5]

Choice of the Exponent *d*. For the power map $x \mapsto x^d$, we consider two natural choices: either choosing $d \geq 3$ as the smallest integer satisfying $\gcd(d, 2^n - 1) = 1$, or $d = -1$, corresponding to the inverse map. For efficiency, we restrict our attention to fields $\mathbb{F}_{2^n}$ where $n = 2^k$ is a power of two. As shown in [GKK+26], the following results hold:

Lemma 1. *For each $k \geq 2$, $2^{2^k} - 1$ is divisible by 3 and by 5. Moreover, $2^{2^k} - 1$ is never divisible by 7.*

Lemma 2. *For each $k \geq 2$ and $0 < d < 2^k$, $2^{2^k} - 1$, then $\gcd(2^{2^k} - 1, 2^d + 1) \neq 1$.*

For $n = 2^k$, the candidates thus reduce to $d = 7$ and $d = -1$. We compare their hardware efficiency and ZK performance to guide our design choice.

From the hardware (HW) point of view, as shown in Tables 2 and 3, $x \mapsto x^{-1}$ is more competitive across relevant field sizes when using binary tower fields. Specifically, the concrete area consumption in terms of lookup tables (LUTs) is between 1.19× and 1.92× lower in the $x \mapsto x^{-1}$ case. Therefore, we choose $x \mapsto x^{-1}$ for our nonlinear layer. The benefit of $x \mapsto x^{-1}$ in our construction is attributed to the efficient tower field inversion [FP97]. However, in general binary extension fields without a tower structure, field inversion is significantly more complex, making $x \mapsto x^7$ the more efficient choice in such constructions. Note that we consider only the HW area consumption here, since the operation latency is largely hidden by pipelining in throughput-oriented designs.

From the Zero-Knowledge (ZK) point of view, as shown in [BBB+26], permutations with low-degree VOLE-ZK multiplicative circuit representation, like SHAKE256, enjoy

[5] We recall that RainHash2.0 is a hash function, its inverse permutation is never evaluated by the user; unlike in a block cipher, there is therefore no need to consider $x \mapsto x^{1/d}$ for decryption.

higher degree *QuickSilver* check optimization allowing several permutation rounds to be checked at once instead of checking each round separately, reducing the proof size. In SHAKE256 S-box, the forward and backward degrees are 2 and 3 respectively, allowing degree-16 VOLE checks with 4 rounds forward and 2 rounds backward. However, when using $x \mapsto x^7$ S-box, such benefits vanish due to already high degree in a single round, moreover, $x \mapsto x^7$ S-box requires 4 multiplications before the VOLE check (higher ZK runtime), compared to $x \mapsto x^{-1}$ S-box performing only an inverse check (Section 6.2).

For the Binius proof system, the respective constraint for the S-box $x \mapsto x^7$ takes x and $y = x^7$ as inputs and proves for the concrete inputs $x^7 + y = 0$. The number of multiplications can again be reduced by introducing the intermediate values x^2, x^4, and x^6, resulting in 4 multiplications. The inversion function x^{-1} is proven by the constraint $(x \cdot y + 1) + (x + \beta y) = 0$, which proves the correct computation of $y = x^{-1}$ from x. The construction of the constraint is further elaborated in Section 6.1. Thus, the $x \mapsto x^7$ S-box consists of 4 multiplications and one addition. The $x \mapsto x^{-1}$ S-box consists of 2 multiplications and 3 additions. Since multiplications are significantly more expensive than additions in the proof system, this gives a slight advantage in terms of constraint costs to the S-box $x \mapsto x^{-1}$ compared to $x \mapsto x^7$.

4.3 About the Linear Layer: LS and PRESENT-Like Designs

Finally, we discuss the rationale behind the linear layer. For efficient software and hardware implementations, we use a lightweight byte-level shuffle, similar to the bit permutation employed in PRESENT. As shown below, this choice does not compromise security.

Wide-Trail Design Strategy. The linear layer is essential for achieving full diffusion in an SPN scheme. Moreover, its diffusion properties can significantly contribute to resistance against statistical attacks, such as differential and the linear attacks, as formalized by the *wide-trail strategy* [DR02a]: instead of using large S-boxes with very low maximum differential probability/linear correlation, this strategy combines relatively small S-boxes with a strongly diffusive linear layer designed to maximize the minimum number of active S-boxes. Consequently, resistance against statistical attacks can be achieved within a few rounds, even with relatively small S-boxes (e.g., 4 or 8 bits). Besides the AES and AES-like designs, several AO primitives [GLR+20, AAB+20, GKR+21, GOPS22, BBC+23] employ linear layers with a high branch number [DR02b], in particular *Maximum Distance Separable* (MDS) matrices, thereby guaranteeing a large minimum number of active S-boxes across two consecutive rounds.[6] This choice is typically justified by the fact that linear operations are considerably cheaper than nonlinear operations in applications such as ZK protocols and MPC.

However, the security benefit of a high-branch-number layer such as an MDS matrix is questionable once the S-box itself is large (e.g., 64 or 128 bits) and strong (very low maximum differential probability $\mathrm{DP}_{\max}$). The wide-trail strategy compensates for small S-boxes by forcing many active S-boxes, since a differential trail has probability roughly $(\mathrm{DP}_{\max})^s$ in the number s of active S-boxes. When $\mathrm{DP}_{\max}$ is already very small – as for the 64-bit inverse S-box of RainHash2.0, where $\mathrm{DP}_{\max} = 2^{-62}$ – even a single active S-box drives a trail quickly below the security bound, so maximizing the number of active S-boxes yields little additional security against single-trail differential and linear cryptanalysis. Choosing an expensive MDS matrix is therefore not ideal when the goal is efficiency in both software and hardware, as for RainHash2.0. We thus adopt a lighter linear layer that still ensures full diffusion of the state, which remains necessary against algebraic and other structural attacks, while avoiding the cost of maximal branch number, as outlined below.

[6] We recall that the branch number of $M \in \mathbb{F}_q^{t \times t}$ for $q = p^n$ is defined as $\mathsf{B}(M) = \min_{x \in \mathbb{F}_q^t \setminus \{0\}} \{\mathrm{hw}(x) + \mathrm{hw}(M \times x)\}$, where $\mathrm{hw}(\cdot)$ is the bundle weight in wide trail terminology.

Design Rationale of L. For a cheap linear layer, we take inspiration from PRESENT [BKL+07], where the linear layer is defined as a bit shuffle: since its S-box operates over only 4 bits, diffusion must be achieved by permuting individual bits. In our case, the S-box is defined over 64 bits, so we can instead shuffle at the byte level, achieving comparable diffusion at a coarser and cheaper granularity. To keep this shuffle simple and elegant, we adopt the LS-design structure [GLSV15], in which a linear layer acts on the rows of the state and the S-box on its columns. For the linear row operation, we use a map similar to ShiftRows of the AES [DR02a]; unlike the AES, however, we deliberately omit a MixColumns-style MDS step, in line with the argument above.

In the AES's ShiftRows, the i-th row is rotated by i positions, achieving full diffusion for a state of 8 rows and at most 8 columns: after one application, every byte has been spread across all columns. However, it is *not* possible to achieve full diffusion in the case of 9 or more columns, since a single ShiftRows moves each byte by at most 8 positions and thus cannot reach every column of a wider state. To recover full diffusion within a single round, we apply ShiftRows twice, once rotating left and once rotating right, and sum the two results. A scaling factor α is applied to one of the two rotations to ensure that the resulting map is invertible, see Proposition 1. Importantly, this lightweight linear layer does not compromise the security of RainHash2.0, as shown in Section 5.

Proposition 1. *Let* L *be the linear layer defined in Equation (8) for* $\alpha \neq 0$*. If* $\mathrm{ord}(\alpha) \nmid t$*, then* L *is invertible.*

Proof. The right and the left rotations correspond to the multiplication with the circulant matrices $C = \mathrm{circ}(0, 0, \ldots, 0, 1) \in \mathbb{F}_{2^8}^{t\times t}$ and $C^{-1} := \mathrm{circ}(0, 1, 0, \ldots, 0) \in \mathbb{F}_{2^8}^{t\times t}$, respectively. In particular, $x \ggg i = C^i \times x$ and $x \lll i = C^{-i} \times x$. Thus, for $i \in \{0, 1, \ldots, 7\}$, the i-th row $x \in \mathbb{F}_{2^8}^t$ is transformed as $x \mapsto \left(C^i + \alpha \cdot C^{-i}\right) \times x$. Let $\mathsf{L}_i := C^i + \alpha \cdot C^{-i} = C^{-1} \cdot \left(C^{2i} + \alpha \cdot I\right)$, where $I \in \mathbb{F}_{2^8}^{t\times t}$ is the identity matrix. Then L is invertible if and only if L_i is invertible for all $i \in \{0, 1, \ldots, 7\}$. Since $\det(C) = \det(C^{-1}) = 1$,

$$\det(\mathsf{L}_i) = 0 \quad \Leftrightarrow \quad \det\left(C^{2i} + \alpha \cdot I\right) = 0 \quad \Leftrightarrow \quad \exists\, x \neq 0 \,:\, \left(C^{2i} + \alpha \cdot I\right) \times x = 0\,.$$

In other words, the values of α for which L_i is not invertible are exactly the (negative) eigenvalues of C^{2i}. Let λ be an eigenvalue of C^{2i} and v the corresponding eigenvector. Then

$$C^{2i} \times v = \lambda v \;\Rightarrow\; (C^{2i})^2 \times v = C^{2i} \times \lambda v = \lambda^2 v \;\Rightarrow\; \ldots \;\Rightarrow\; (C^{2i})^t \times v = \lambda^t v\,.$$

Since $(C^{2i})^t = (C^t)^{2i} = I^{2i} = I$, it follows that all eigenvalues of C^{2i} are t-th roots of unity, i.e., $\lambda^t = 1$, and thus $\mathrm{ord}(\lambda) \mid t$ by Lagrange's theorem. □

Remark 2. For $\mathbb{F}_{2^8} := \mathbb{F}_2[X]/\langle \mathsf{I} \rangle$, using $\alpha = X \mod \mathsf{I}$ in Equation (8) yields an invertible linear layer L for common choices of field moduli I:

- For the modulus $\mathsf{I} = X^8 + X^4 + X^3 + X + 1$ used in AES, $\mathrm{ord}(\alpha) = 51 \nmid t$.
- For the modulus $\mathsf{I} = X^8 + X^4 + X^3 + X^2 + 1$, $\mathrm{ord}(\alpha) = 255 \nmid t$.

In the tower field setting, one may also choose the elements of $\mathbb{T}_3$ that are isomorphic to the above choices; they have the same multiplicative order in the tower representation. However, a more hardware-friendly representative is $\alpha = x_2 \equiv 4$ since $\mathrm{ord}(4) = 5 \nmid t$.

5 Security Analysis

In this section, we present our security analysis of RainHash2.0. We show that 8 rounds are largely sufficient to prevent algebraic and statistical attacks. In more details, statistical

attacks are prevented after at most 6 rounds: differential and linear attacks are infeasible beyond 5 rounds (Section 5.1.1), truncated differentials beyond 3 rounds (Section 5.1.2), and impossible differentials beyond 2 rounds (Section 5.1.3). Regarding algebraic attacks, we consider several modeling strategies (over $\mathbb{F}_{2^{64}}$, $\mathbb{F}_{2^8}$, and $\mathbb{F}_2$): in all of them, the polynomial representation reaches its maximum degree and full density within 5 rounds, preventing interpolation attacks beyond 4 rounds (Section 5.2.3), and Gröbner basis attacks do not extend beyond 4 rounds either (Section 5.2.4). The strongest attack we found is a rebound attack covering 6 rounds (Section 5.3). Adding a security margin of 2 rounds, we obtain 8 rounds in total, for both $t = 8$ and $t = 12$. We consider this margin appropriate: only the rebound attack covers 6 rounds, while all other attacks stop at 5 rounds or fewer. In particular, it is deliberately conservative with respect to algebraic attacks, an area under active development in which new techniques regularly emerge.

5.1 Statistical Attacks

We start by presenting the security analysis of RainHash2.0 against statistical attacks.

5.1.1 Differential and Linear Attacks

Differential Attack. In a differential attack [BS91], the idea is to study how differences between two inputs propagate through a symmetric primitive. Notably, if a specific input difference Δ_{in} leads to a specific output difference Δ_{out} with probability higher than that of a random permutation, this property can be exploited by an attacker. For RainHash2.0, this probability is given by

$$\text{Prob}(\Delta_{\text{in}} \to \Delta_{\text{out}}) = \frac{|\{x \in \mathbb{F}_{2^8}^{8\times t} \mid \mathsf{P}(x \oplus \Delta_{\text{in}}) \oplus \mathsf{P}(x) = \Delta_{\text{out}}\}|}{2^{64\cdot t}}.$$

To argue resistance of RainHash2.0 against differential attacks, we compute lower bounds on the minimal number s of active S-boxes. Under the Markov cipher assumption and assuming independence of the rounds, it holds that

$$\text{Prob}(\Delta_{\text{in}} \to \Delta_{\text{out}}) \leq (\text{DP}_{\max})^s \quad \forall\, \Delta_{\text{in}}, \Delta_{\text{out}} \in \mathbb{F}_{2^8}^{8\times t},$$

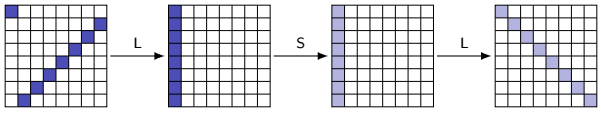
where $\text{DP}_{\max}$ denotes the maximum differential probability of the S-box. In particular, the RainHash2.0 S-box satisfies the following properties:

- $\text{DP}_{\max}(x \mapsto x^{-1}) = 4/2^{64} = 2^{-62}$;
- at least one S-box is active per round, i.e., at least r S-boxes are active over r rounds, independently of the value of t.

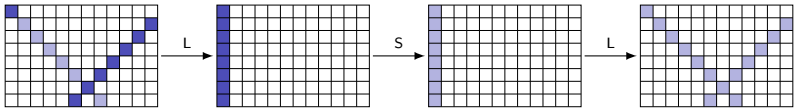
Hence, for all $\Delta_{\text{in}}, \Delta_{\text{out}} \in \mathbb{F}_{2^8}^{8\times t}$, $\text{Prob}(\Delta_{\text{in}} \to \Delta_{\text{out}}) \leq 2^{-62\,\cdot\, r}$. In particular, for $r \geq 6$, this bound is below $2^{-2.5\cdot 128}$, i.e., 2.5× the target security level of 128 bits. The factor of 2.5× accounts for both the clustering effect (i.e., the combination of multiple differential characteristics) and the potential use of meet-in-the-middle techniques.

Linear Attack. In a linear attack [Mat94], the attacker exploits the existence of linear approximations of the primitives that holds with a probability different than in the case of a random permutation. As in the majority of the cases, security against differential cryptanalysis implies security against linear cryptanalysis as well. Here, we limit ourselves to study in more details the case in which the state $x \in \mathbb{F}_{2^8}^{8\times t}$ is defined as

$$x' \otimes \hat{x} = \begin{bmatrix} x'_0 \cdot \hat{x}_0 & x'_0 \cdot \hat{x}_1 & x'_0 \cdot \hat{x}_2 & \dots & x'_0 \cdot \hat{x}_{t-1} \\ x'_1 \cdot \hat{x}_0 & x'_1 \cdot \hat{x}_1 & x'_1 \cdot \hat{x}_2 & \dots & x'_1 \cdot \hat{x}_{t-1} \\ \vdots & & & \ddots & \vdots \\ x'_7 \cdot \hat{x}_0 & x'_7 \cdot \hat{x}_1 & x'_7 \cdot \hat{x}_2 & \dots & x'_7 \cdot \hat{x}_{t-1} \end{bmatrix}$$



(a) $t = 8$ with $\alpha = 0$.



(b) $t = 12$ with $\alpha \neq 0$.

Figure 5: Truncated differential for 1-round RainHash2.0. A byte with non-zero difference is denoted in blue.

for $x' \in \mathbb{F}_{2^8}^8$ and $\hat{x} \in \mathbb{F}_2^t$. In such a case, we have that

$$\mathsf{S}(x' \otimes \hat{x}) = \mathsf{S}(x') \otimes \hat{x}\,, \qquad \mathsf{L}(x' \otimes \hat{x}) = x' \otimes \mathsf{L}(\hat{x})\,.$$

As showed in [GLSV15, Section 4.1], this implies the same behavior for differences and for linear masks, in the sense that it is possible to build differential characteristics (resp. linear trails) where the differences (resp. selection masks) are written as tensor products. In particular, we have that $\delta_{\text{in}} \times \hat{x} \to \delta_{\text{out}} \times \mathsf{L}(\hat{x})$ through one round with probability $\leq 2^{-62 \cdot |\hat{x}|}$, where $|\hat{x}|$ denotes the Hamming weight of $\hat{x}$.

5.1.2 Truncated Differential Attacks

Truncated differential attacks [Knu95] are a variant of differential attacks in which the attacker only partially knows the difference between two states (usually, the fact that the difference is zero or non-zero). In the following, we omit for simplicity the initial linear layer.

1-Round. A truncated differential with probability 1 can always be set up for 1 round, as shown in Figure 5 for $t = 8$ and $t = 12$. No truncated differential with probability 1 exists for 2 or more rounds, since full diffusion is achieved within 1 round.

2-Rounds. When working over 2 rounds, a truncated differential with probability

$$2^{-8 \cdot (8-1)} = 2^{-56}$$

exists, as shown in Figure 8 in App. A.1 for $t = 8$ and $t = 12$. Before the last linear layer, note that the probability for the random permutation case is $2^{-64 \cdot (t-1)} \leq 2^{-448}$, hence, much smaller. As it is possible to see in there, this is related to the probability that the S-box maps the input difference into one in which only the first byte is active.

We limit ourselves to point out that, if the position of the non-zero byte difference is not fixed, then the probability just given increases by a factor 8.

3-Rounds. By iterating a strategy similar to the one just described, it is possible to set up a 3-round truncated differential with probability $2^{-2 \cdot 56} = 2^{-112}$, as shown in Figure 9 in App. A.1 for $t = 8$ and $t = 12$.

5.1.3 Other Statistical Attacks

Impossible differential attacks [BBS99] exploit the existence of (truncated) differential characteristics that hold with probability 0. They are usually set up by concatenating truncated differentials that hold with probability 1 in such a way to have a contradiction in the middle. Due to the previous results, impossible differentials exist up to 2 rounds, and a similar conclusion holds for zero-linear correlation [BR14].

Finally, especially due to the weak arrangement of our design, we limit ourselves to mentioning that other statistical attacks, including integral attacks [DKR97], rotational cryptanalysis [KN10], slide attacks [BW99], multiple-of-8 [GRR17], mixture differential cryptanalysis [Gra18], among others, are not a threat to our design.

5.2 Algebraic Attacks

Algebraic attacks model the primitive in a concrete attack scenario as a system of polynomial equations, and exploit low degree or sparsity to solve for the unknowns (e.g., a preimage given its hash). Because RainHash2.0 combines functions over two different fields, its resistance to these attacks depends on how its description translates between the base field and its extension, where the degree and density of the resulting polynomial system play a crucial role. The remainder of this section is organized as follows. In Section 5.2.1, we recall (fake) Weil descent, a general technique for switching between representations over the base and extension fields. In Section 5.2.2, we apply it to derive the polynomial representations of the individual nonlinear and linear components of RainHash2.0. Building on these, we study the degree and density of the resulting system, which govern the cost of interpolation attacks (Section 5.2.3), and finally the complexity of solving the full polynomial system via Gröbner bases (Section 5.2.4). Detailed experimental results are given in App. A.2 to A.4.

Notation. Throughout this section, we work with a finite field $F := \mathbb{F}_q$ for a prime power $q = p^n$, and its degree-m extension $F_m := \mathbb{F}_{q^m}$, which forms an m-dimensional F-vector space F_m/F. Given an F-basis $\mathcal{B} = \{\beta_0, \ldots, \beta_{m-1}\}$ of F_m, any $X \in F_m$ can be uniquely written as $X = \sum_{j=0}^{m-1} x_j \beta_j$ with $x_j \in F$, and $\mathcal{B}$ induces an F-linear isomorphism $\Phi : F^m \to F_m$, $(x_0, \ldots, x_{m-1}) \mapsto \sum_{j=0}^{m-1} x_j \beta_j$, whose inverse gives the coordinate vector of X with respect to $\mathcal{B}$. Here, we use uppercase letters (e.g., X) for elements of the extension field F_m, and lowercase letters (e.g., x_j) for elements of the base field F. In the binary case, F and F_m are extensions of $\mathbb{F}_2$ of degrees $n_1 = [F : \mathbb{F}_2] = 2^{k_1}$ and $n_2 = [F_m : \mathbb{F}_2] = 2^{k_2}$ (so $q_1 = |F| = 2^{n_1}$, $q_2 = |F_m| = 2^{n_2}$, and $m = 2^{k_2 - k_1}$), giving the tower $\mathbb{F}_2 \subset F \subset F_m$.

Remark 3. For RainHash2.0, the linear layer operates over F and the nonlinear layer over the extension F_m. We deliberately keep this two-field setting general to allow flexible instantiation in the algebraic models and experiments. For our concrete instances, we use $k_1 = 3$ and $k_2 = 6$, yielding $F \cong \mathbb{F}_{2^8}$ and $F_m \cong \mathbb{F}_{2^{64}}$ with $m = 8$. In practice, these fields are realized using the binary tower fields $\mathbb{T}_{k_1}$ and $\mathbb{T}_{k_2}$, as defined in Section 2.1.

5.2.1 Weil and Fake Weil Descent

Let $\mathsf{F} \in F_m[X]$ be a polynomial over the extension field F_m/F. Writing $X = \sum_{j=0}^{m-1} x_j \beta_j$ in the F-basis $\mathcal{B} = \{\beta_0, \ldots, \beta_{m-1}\}$ and substituting into F yields a decomposition of F into a linear combination of the basis elements with coefficients $[\mathsf{F}]_j \in F[x_0, \ldots, x_{m-1}]$:

$$\mathsf{F}(X) = \mathsf{F}\left(\sum_{j=0}^{m-1} x_j \beta_j\right) \equiv \sum_{j=0}^{m-1} [\mathsf{F}]_j \beta_j \pmod{(x_0^q - x_0, \ldots, x_{m-1}^q - x_{m-1})},$$

where $\deg_{x_j}([\mathsf{F}]_j) < q$ for all j. In particular, a polynomial system over F_m can be transformed into a system over the base field F with m times as many variables.

Definition 2 (Weil Descent, [HKY15])**.** Let $\mathcal{F} \subset F_m[X]$ be a set of polynomials. The *Weil descent system* of $\mathcal{F}$ with respect to the F-basis $\mathcal{B}$ is defined as

$$\mathcal{F}' = \{[\mathsf{F}]_j : \mathsf{F} \in \mathcal{F},\ 0 \le j < m\} \ \cup\ \{x_j^q - x_j : 0 \le j < m\}\,.$$

The solutions of $\mathcal{F}$ over F_m correspond bijectively to the solutions of $\mathcal{F}'$ over F via the coordinate map $X = \sum_{j=0}^{m-1} x_j\beta_j$.

An important example arises from F-linear maps $\mathsf{L} : F_m \to F_m$, which correspond to linear transformations of the coordinate vector $\mathbf{x} = (x_0, \ldots, x_{m-1}) \in F^m$.

Example 1 (F-linear maps)**.** Every F-linear map $\mathsf{L} : F_m \to F_m$ can be written uniquely as a q-linearized polynomial $\mathsf{L}(X) = \sum_{i=0}^{m-1} c_i X^{q^i}$ with coefficients $c_i \in F_m$. Substituting $X = \sum_{j=0}^{m-1} x_j\beta_j$ into $\mathsf{L}(X)$ and using $x_j^{q^i} = x_j$ for $x_j \in F$ yields

$$\mathsf{L}(X) = \sum_{i=0}^{m-1} c_i \cdot \left(\sum_{j=0}^{m-1} x_j\beta_j\right)^{q^i} = \sum_{j=0}^{m-1} \left(\sum_{i=0}^{m-1} c_i\beta_j^{q^i}\right) x_j\,.$$

Decomposing the coefficients $\sum_{i=0}^{m-1} c_i\beta_j^{q^i}$ with respect to the F-basis $\mathcal{B}$ yields

$$\mathsf{L}(X) = \sum_{j=0}^{m-1} \left(\sum_{k=0}^{m-1} a_{k,j}\beta_k\right) x_j = \sum_{k=0}^{m-1} \left(\sum_{j=0}^{m-1} a_{k,j}x_j\right) \beta_k$$

with $a_{k,j} \in F$. The coefficients of the basis elements β_k give $y_k = \sum_{j=0}^{m-1} a_{k,j}x_j$ for $k \in \{0, \ldots, m-1\}$, which corresponds to the linear transformation $\mathbf{y} = A\mathbf{x}$, with $A \in F^{m\times m}$.

While Weil descent expands elements of F_m in an F-basis, the so-called fake Weil descent instead expands the exponent structure of monomials, allowing a univariate polynomial $\mathsf{F} \in F_m[X]$ to be rewritten as a multivariate polynomial $\overline{\mathsf{F}} \in F_m[X_0, \ldots, X_{m-1}]$. For $e \in \mathbb{Z}_{\ge 0}$, let $X^{e'}$ be the remainder of X^e modulo $X^{q^m} - X$ in $F_m[X]$ and write $e' = \sum_{j=0}^{m-1} e'_j q^j$ in base q with $e'_j \in \{0, \ldots, q-1\}$. Then

$$X^{e'} = \prod_{j=0}^{m-1} \left(X^{q^j}\right)^{e'_j}\,.$$

Substituting $X_j := X^{q^j}$ yields $\overline{X^e} = X_0^{e'_0} \cdots X_{m-1}^{e'_{m-1}}$. This map can be extended F_m-linearly to all polynomials in $F_m[X]$, yielding a map $\overline{(\cdot)} : F_m[X] \to F_m[X_0, \ldots, X_{m-1}]$.

Definition 3 (Fake Weil Descent, [HKY15])**.** Let $\mathcal{F} \subset F_m[X]$ be a set of polynomials. The *fake Weil descent system* of $\mathcal{F}$ is defined as

$$\overline{\mathcal{F}} = \{\overline{\mathsf{F}} : \mathsf{F} \in \mathcal{F}\} \cup \{X_0^q - X_1, \ldots, X_{m-1}^q - X_0\}\,.$$

There is a bijection between the set of solutions of $\mathcal{F}$ and $\overline{\mathcal{F}}$ over F_m.

Fake Weil descent systems can be used, for example, to study the algebraic structure of nonlinear maps over F_m or to facilitate the computation of affine linearized polynomials.

Example 2 (Inversion map over $\mathbb{F}_{2^m}$)**.** Consider the inversion map $X \mapsto Y = X^{-1}$, which satisfies $X^{-1} = X^{2^m-2}$ for $X \neq 0$. Since $2^m - 2 = \sum_{j=1}^{m-1} 2^j$, we obtain $X^{2^m-2} = \prod_{j=1}^{m-1} X^{2^j}$. Introducing variables $X_j := X^{2^j}$ for $j = 0, \ldots, m-1$, the fake Weil descent yields a multivariate polynomial system over F_m modeling the inversion map

$$\{X_1X_2\cdots X_{m-1} - Y\} \ \cup\ \{X_0^2 - X_1, X_1^2 - X_2, \ldots, X_{m-1}^2 - X_0\}\,.$$

Table 4: Systems of polynomial equations representing the coordinate functions of the nonlinear and linear layers – $\mathsf{S}_j : F_m \to F_m$, $\mathsf{L}_i : F^t \to F^t$, $\mathsf{L}'_j : F_m^t \to F_m$ for $0 \le j < t$ and $0 \le i < m$ – over different fields F_m, F, and $\mathbb{F}_2$. Monomial counts include both input and output variable terms.

Layer	Model	Field	#Eq	#InVar	#OutVar	Max. degree d	Max. mon./eq
S_j	$Y = X^{-1}$	F_m	1	1	1	$q_2 - 2$	2
S_j	$Y = X^{-1}$	F	$m = 2^{k_2-k_1}$	m	m	$m(q_1-1)-1$	$q_1^m = q_2$ †
S_j	$Y = X^{-1}$	$\mathbb{F}_2$	$n_2 = 2^{k_2}$	n_2	n_2	$n_2 - 1$	q_2
S_j	$XY = 1$	F_m	1	1	1	2	2
S_j	$XY = 1$	F	$m = 2^{k_2-k_1}$	m	m	2	m^2+1
S_j	$XY = 1$	$\mathbb{F}_2$	$n_2 = 2^{k_2}$	n_2	n_2	2	n_2^2+1
L'_j	lin. poly.	F_m	1	t	1	q_1^{m-1}	$mt+1$
L_i	matrix	F	t	t	t	1	$t+1$
L_i	matrix	$\mathbb{F}_2$	$tn_1 = t2^{k_1}$	tn_1	tn_1	1	tn_1+1

† For $m = 2$, a tighter bound is given by $2q_1 - 1$.

Notably, fake Weil descent does not reduce the coefficient field from F_m to F. Instead, it merely changes the representation. However, for cryptanalysis, the variables X_j can subsequently be expanded in an F-basis, yielding a representation over F (see, e.g., Section 5.2.2).

5.2.2 Polynomial Representation of the Individual Components

Algebraic attacks can be more efficient when the polynomial representation of the cipher is sparse or of low degree. We thus investigate in the following the degree and density of different polynomial representations of $\mathsf{S}_j : F_m \to F_m$ and $\mathsf{L}_i : F^t \to F^t$, $\mathsf{L}'_j : F_m^t \to F_m$ – the coordinate functions of the nonlinear and linear layers – over F_m, F, and $\mathbb{F}_2$, summarized in Table 4.

Polynomial Representation of the Nonlinear Layer ($Y = X^{-1}$). The inversion map is directly expressed as the univariate polynomial $Y = X^{-1} = X^{q_2-2}$ over F_m, yielding a single equation of degree $q_2 - 2$ in two variables. We use fake Weil descent [HKY15] to obtain coordinate representations over F and $\mathbb{F}_2$.

- *Over F.* Since the exponent expands as $q_2 - 2 = q_1^m - 2 = (q_1 - 2) + \sum_{j=1}^{m-1}(q_1 - 1)q_1^j$ in base q_1, we obtain

$$Y = X^{-1} = X^{q_1-2} \prod_{j=1}^{m-1} \left(X^{q_1{}^j}\right)^{q_1-1} = X_0^{q_1-2} \prod_{j=1}^{m-1} X_j{}^{q_1-1} \tag{13}$$

  where $X_j := X^{q_1^j}$, satisfying $X_{j+1} = X_j^{q_1}$ for $0 \le j < m$ and $X_m = X_0$. Writing $X = \sum_{i=0}^{m-1} x_i\beta_i$ with $x_i \in F$, and noting that $x_i^{q_1} = x_i$, each X_j is an F-linear combination of the same variables x_i:

$$X_j = X^{q_1^j} = \left(\sum_{i=0}^{m-1} x_i\beta_i\right)^{q_1^j} = \sum_{i=0}^{m-1} x_i\beta_i^{q_1^j}. \tag{14}$$

  Substituting each X_j in Y by an F-linear form in $x_0, \ldots, x_{m-1}$ and expanding with respect to the basis $\mathcal{B}$ yields coordinate polynomials $y_i = \mathsf{F}_i(x_0, \ldots, x_{m-1}) \in F[x_0, \ldots, x_{m-1}]$ for $0 \le i < m$, each of total degree at most $d = m(q_1 - 1) - 1$ with at most $\binom{m+d}{d}$ monomials. More precisely, counting all monomials in m variables

with each exponent in $\{0, \ldots, q_1 - 1\}$ – minus the single monomial $x_0^{q_1-1} \cdots, x_{m-1}^{q_1-1}$ which has total degree $m(q_1 - 1) > d$, plus the single monomial y_i – yields a total of $q_1^m = q_2$ monomials per constraint.[7]

- *Over* $\mathbb{F}_2$. Expanding X and Y in an $\mathbb{F}_2$-basis $\{\gamma_0, \ldots, \gamma_{n_2-1}\}$ of F_m and substituting into $Y = X^{q_2-2}$ yields n_2 coordinate equations. Since the variables are in $\mathbb{F}_2$, every polynomial reduces to multilinear form, giving degree at most $n_2 - 1$ with at most q_2 monomials per constraint.

Polynomial Representation of the Nonlinear Layer ($X \cdot Y = 1$). The inversion relation is expressed as the single bilinear equation $XY = 1$ over F_m, yielding one equation of degree 2 in two variables. For coordinate representations over F and $\mathbb{F}_2$, we use Weil descent.

- *Over* F. Writing $X = \sum_{i=0}^{m-1} x_i\beta_i$ and $Y = \sum_{i=0}^{m-1} y_i\beta_i$ with $x_i, y_i \in F$ and expanding the product yields

$$XY = \left(\sum_{i=0}^{m-1} x_i\beta_i\right)\left(\sum_{j=0}^{m-1} y_j\beta_j\right) = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} x_i y_j\,(\beta_i\beta_j)\,. \tag{15}$$

  Expanding each basis product as $\beta_i\beta_j = \sum_{k=0}^{m-1} c_{i,j,k}\beta_k$ with $c_{i,j,k} \in F$ and writing $1 = \sum_{k=0}^{m-1} d_k\beta_k$, comparing coefficients of β_k yields m bilinear equations over F in $2m$ variables:

$$\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} c_{i,j,k}\, x_i y_j = d_k\,, \quad 0 \le k < m\,. \tag{16}$$

  Each equation has degree 2 and at most $m^2 + 1$ monomials.

- *Over* $\mathbb{F}_2$. Applying the same approach as over F using the $\mathbb{F}_2$-basis $\{\gamma_0, \ldots, \gamma_{n_2-1}\}$ of F_m yields n_2 bilinear equations in $2n_2$ variables, each of degree 2 with at most $n_2^2 + 1$ monomials.

The degree and density of the coordinate polynomials modeling the nonlinear layer over $F = \mathbb{T}_{k_1}$ for different tower fields are reported in Table 12 in App. A.2. Notably, for the model $Y = X^{-1}$, the monomial bound depends on both q_1 and m: for fixed m, it grows rapidly with q_1 while the actual monomial counts grow far more slowly, so the polynomials become increasingly sparse relative to the bound. For the model $XY = 1$, the bound $m^2 + 1$ depends only on m, and apart from the base case $k_1 = 0$, the monomial counts are consistent across tower levels for m fixed. As expected due to the tower field construction, the average density decays gradually with m, reaching around 30% for the suggested instance $(k_2, k_1) = (6, 3)$. A key practical advantage of $XY = 1$ is tractability: coordinate polynomials can be computed for all instances considered, whereas the $Y = X^{-1}$ model did not complete in reasonable time for larger fields.

Polynomial Representation of the Linear Layer. The linear layer $\mathsf{L} : F^{m\times t} \to F^{m\times t}$ is F-linear and naturally represented as a matrix-vector product over F. We consider two coordinate views depending on the target field.

[7] Notably, for $m = 2$, the norm factorization $X^{-1} = \frac{X^{q_1}}{N(X)}$ with $N(X) = X^{q_1+1} \in F$ reduces the bound to at most $2q_1 - 1$ monomials per constraint, far below the general bound q_1^2. Similar norm-based factorizations apply recursively for larger $m = 2^{k_2-k_1}$, but the advantage weakens with each level and a closed-form bound becomes increasingly involved.

- *Over F_m.* Composing each column into a single F_m element yields the coordinate function $\mathsf{L}'_j : F_m^t \to F_m$ for $0 \le j < t$, mapping the t input columns to the j-th output column. Since L is F-linear, each L'_j admits an F-linearized polynomial representation over F_m, i.e., a linear combination of Frobenius powers:

$$\mathsf{L}'_j : F_m^t \to F_m\,, \quad (X_0, \ldots, X_{t-1}) \mapsto Y_j = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k} X_i^{q_1^k}\,, \tag{17}$$

  where $C_{j,i,k} \in F_m$. This yields 1 equation of degree q_1^{m-1} with at most $mt+1$ monomials.

- *Over F.* The linear layer naturally acts row by row via $\mathsf{L}_i : F^t \to F^t$, see Equation (8). This yields t linear equations in $2t$ variables over F, with at most $t+1$ monomials per equation.

- *Over $\mathbb{F}_2$.* Expanding each F-element in row k into its $\mathbb{F}_2$-basis $\{\gamma_0, \ldots, \gamma_{n_1-1}\}$ of F yields tn_1 linear equations in $2tn_1$ variables over $\mathbb{F}_2$, with at most tn_1+1 monomials per equation.

Table 13 in App. A.2 lists the coefficients of L'_0 for $F = \mathbb{T}_3$, $F_m = \mathbb{T}_6$, computed via linear algebra. We observe that the representation is dense, with each coefficient occurring at most four times in both cases $t = 8$ and $t = 12$. This behavior extends to all L'_j, where the same set of coefficients appears, differing only in their ordering.

5.2.3 Interpolation Attacks: Degree and Density

Interpolation attacks [JK97] aim to exploit the low degree properties by reconstructing the polynomial that describes the permutation from input/output pairs by means of Lagrange interpolation. The complexity of this attack is dominated by the degree and number of monomials in the polynomial representation.

Growth of the Degree. The degree of $Y = X^{-1}$ is $d_2 = q_2 - 2$ over F_m. For $\mathsf{L} \circ \mathsf{S}$, the composite function for the j-th coordinate of the state over F_m is

$$Y_j = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k} \left(X_i^{q_2-2}\right)^{q_1^k} = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k} X_i^{q_2-1-q_1^k}\,. \tag{18}$$

Since $C_{j,i,k} \neq 0$ when $k = 0$, the degree of $\mathsf{L} \circ \mathsf{S}$ is also d_2. To reach the maximum degree, the total number of rounds R should satisfy $d_2^{\mathrm{R}} > t \cdot (d_2 + 1) - 1$.

When considering polynomial representation over F, the inversion mapping $Y = X^{-1}$ over F_2 is decomposed into m coordinate polynomials via fake Weil descent. The total degree of round function is $d_1 = m(q_1 - 1) - 1$. To reach the maximum degree, the total number of rounds R should satisfy $d_1^{\mathrm{R}} > t \cdot (m(q_1 - 1)) - 1$. The algebraic degree of $Y = X^{-1}$ is $\mathrm{hw}(q_2 - 2) = 2^{k_2} - 1 = 63$. To reach the maximum algebraic degree, the total number of rounds should satisfy $(63)^{\mathrm{R}} \ge n_2 \cdot t$. In total,

$$\mathrm{R} \ge \max\{\log_{d_2}\left(t(d_2+1)-1\right), \log_{d_1}\left(t(d_1+1)-1\right), \log_{63}(n_2) + \log_{63}(t)\}\,. \tag{19}$$

For our concrete instances with $(k_1, k_2) = (3, 6)$, the threshold is $\mathrm{R} = 2$ for both $t = 8$ and $t = 12$. In order to avoid the reduced degree of the polynomial representation, we assume that the polynomial representation reaches its maximum degree after 3 rounds.

Density. Even if the degree reaches its maximum, a sparse polynomial can still be reconstructed efficiently via interpolation. We analyze density in three algebraic models: over F_m with t variables, over F with mt variables, and over $\mathbb{F}_2$ with $n_2 t$ variables. Although the three models describe the same function, the density behavior may differ between representations and an attacker could potentially exploit sparsity in one model even if the other is dense. We adopt the following notion of density.

Definition 4. A polynomial over $\mathbb{F}_q$ with u variables is considered *dense* if it contains $\Omega(q^u)$ monomials.

For a random polynomial over $\mathbb{F}_q$ with u variables, each coefficient is non-zero with probability $q - 1/q$. So the expected number of non-zero monomials is $(q-1)q^{u-1}$. To evaluate this for $t = m$ and $t = 3m/2$ in practice, we perform experimental tests for $(k_1, k_2, t) = (1, 2, 2), (1, 2, 3)$. The results are shown in Figure 10 in App. A.3. In all three models, the number of monomials reaches the expected value within a small number of rounds: R = 4 for F_m, R = 5 for F and $\mathbb{F}_2$. This confirms the density property.

Interpolation using Fraction. Following the approach of [JK97], an attacker can represent the R-round permutation as a fraction P/Q rather than a polynomial to exploit the low degree of the inversion map $Y = 1/X$. Such a representation, where P and Q have $|P|$ and $|Q|$ unknown coefficients respectively, can be determined using $|P|+|Q|$ plaintext-ciphertext pairs.

As we can see, after the initial linear layer we have $Z_j = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k}(X_i)^{q_1^k}$, which has mt monomials and degree q_1^{m-1}. Then after the first round function, the j-th coordinate is

$$Y_j^{(1)} = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k}(\frac{1}{Z_i})^{q_1^k} + c_j^{(1)} = \frac{P_{1,j}(X_0, \cdots, X_{t-1})}{Q_{1,j}(X_0, \cdots, X_{t-1})}.$$

The common denominator is $Q_{1,j}(X_0, \cdots, X_{t-1}) = \prod_{i=0}^{t-1} Z_i^{q_1^{m-1}}$, which has degree $d_1 = t \cdot q_1^{2(m-1)}$ and at most $s_1 = (tm)^t$ monomials. For $P_{1,j}$, the algebraic expression is even more complex with the same degree bound.

After round r, we have

$$Y_j^{(r)} = \sum_{i=0}^{t-1} \sum_{k=0}^{m-1} C_{j,i,k} \cdot \frac{Q_{r-1,i}^{q_1^k}(X_0, \cdots, X_{t-1})}{P_{r-1,i}^{q_1^k}(X_0, \cdots, X_{t-1})} + c_j^{(r)} = \frac{P_{r,i}(X_0, \cdots, X_{t-1})}{Q_{r,i}(X_0, \cdots, X_{t-1})}.$$

As $Q_{r,j} = \prod_{i=0}^{t-1} P_{r-1,i}^{q_1^{m-1}}$, the number of monomials in $Q_{r,j}$ grows exponentially as $s_r \approx s_{r-1}^t$. For our concrete instance $(k_1, k_2) = (3, 6)$, the number of monomials reaches $s_2 \approx (s_1)^t \approx (tm)^{t^2} \gg 2^{128}$.

Combined with the meet-in-the-middle technique, an attacker must solve a linearized system $P_{r_1}\tilde{Q}_{r_2} - \tilde{P}_{r_2}Q_{r_1} = 0$, which requires data proportional to $2 \cdot s_{r_1} \cdot s_{r_2}$. Consequently, the required data complexity surpasses the entire codebook, rendering fractional interpolation attacks impossible after only three rounds.

5.2.4 Polynomial System Solving (PoSSo)

We model RainHash2.0 in the different attack scenarios as a system of polynomial equations by introducing a fresh variable for each S-box output at every round, as well as variables for the (unknown) inputs. The goal is then to solve this system for the unknown variables, typically using Gröbner basis methods. We begin with a general description of the Gröbner basis attack strategy, then detail the concrete attack scenarios arising from the sponge and compression modes, and finally describe the algebraic models over F_m, F, and $\mathbb{F}_2$.

Gröbner Basis Attack. Let $\mathcal{F} = \{\mathsf{F}_1, \ldots, \mathsf{F}_m\} \subset \mathbb{F}_q[x_1, \ldots, x_n]$ be the polynomial system describing the primitive over the finite field $\mathbb{F}_q$ and let $\mathcal{I} = \langle \mathsf{F}_1, \ldots, \mathsf{F}_m \rangle$ denote the ideal generated by $\mathcal{F}$. In the settings considered in symmetric cryptanalysis, the ideal $\mathcal{I}$ is typically zero-dimensional, i.e., the corresponding variety contains finitely many solutions. A Gröbner basis attack usually proceeds in three steps:

1. First, a Gröbner basis of the polynomial system is computed with respect to a suitable monomial ordering, often degree reverse lexicographic, using algorithms such as F4 [Fau99] or F5 [Fau02]. The complexity of this step is estimated as

$$\mathcal{O}\left(\binom{n + D_{\text{sol}}}{D_{\text{sol}}}^{\omega}\right), \tag{20}$$

   where n is the number of variables, D_{sol} is the solving degree of the system, and $2 < \omega \leq 3$ denotes the linear algebra constant.[8]

   The solving degree is commonly estimated by the degree of regularity D_{reg}: for overdetermined systems behaving as semi-regular sequences, D_{reg} is the index of the first non-positive coefficient of the Hilbert series

$$S_{m,n}(z) = \frac{\prod_{i=1}^{m}(1 - z^{\deg(\mathsf{F}_i)})}{(1-z)^n}. \tag{21}$$

   For determined systems forming a regular sequence, $D_{\text{reg}} = 1 + \sum_{i=1}^{m}(\deg(\mathsf{F}_i) - 1)$ exactly. The solving degree and D_{reg} often coincide in practice but may differ by late-occurring syzygies, cf. [BFS04, CG23].

2. When the polynomial system is not defined over $\mathbb{F}_2$, the second step is to extract a univariate polynomial from the system. This is typically done by converting the Gröbner basis to lexicographic order using the FGLM algorithm [FGLM93, FM17], which yields a triangular system and has complexity

$$\mathcal{O}\left(nD_{\mathcal{I}}^{\omega}\right), \tag{22}$$

   where $D_{\mathcal{I}}$ is the degree of the ideal.

   The degree $D_{\mathcal{I}}$ of a zero-dimensional ideal $\mathcal{I}$ is the number of common zeros of its generators in the algebraic closure $\overline{\mathbb{F}}_q$, counted with multiplicities. For determined systems, it is bounded above by the Bézout bound $\prod_{i=1}^{n}\deg(\mathsf{F}_i)$. Notably, if $D_{\mathcal{I}}$ is small, the FGLM conversion step is comparatively cheap.

3. The FGLM conversion yields a Gröbner basis in triangular shape, including a single univariate polynomial of degree $D_{\mathcal{I}}$. Its roots are found in

$$\mathcal{O}\left(D_{\mathcal{I}}\log(D_{\mathcal{I}})(\log(D_{\mathcal{I}}) + \log(q))\log\log(D_{\mathcal{I}})\right) \tag{23}$$

   operations over $\mathbb{F}_q$ [BBLP22], which is typically negligible compared to the Gröbner basis computation. If further solution coordinates are needed, they are obtained by back-substitution through the triangular system.

Concrete Attack Scenarios. We model attacks on the sponge hash function H and the compression function C as polynomial system solving problems over the underlying R-round permutation P, and consider the following scenarios:

[8] The complexity reflects the cost of the linear algebra on the Macaulay matrix at degree d, where d is the highest degree reached during the F4/F5 computation. $\omega = \log_2(7) \approx 2.81$ from Strassen's algorithm [Str69] or the current best bound $\omega \approx 2.37$ [WXXZ23].

Table 5: Constrained input/output words $(k_{\text{in}}, k_{\text{out}})$ for the different attack scenarios, measured in F_m-words. We consider two state sizes: a *square* state with $t = m$ and a *wide* state $t = 3m/2$. In both cases, $c = \chi = m/2$. See Table 1 for the case $F_m = \mathbb{F}_{2^{64}}$.

	Sponge		**Compression**	
	(2nd) preimage	CICO-k for $k \leq m/2$	preimage	2nd preimage
$t = m$	$(m/2,\ m/2)$	(k, k)	$(0,\ m/2)$	$(m/2,\ m/2)$
$t = 3m/2$	$(m/2,\ m/2)$	(k, k)	$(0,\ m/2)$	$(m,\ m/2)$

- For the *sponge mode*, both preimage and second-preimage attacks on a single message block with $c = \chi$ map to the same CICO-c problem: c input words are fixed to zero, $\chi = c$ output words are fixed to the target hash value, and the remaining $r = t - c$ input words remain as unknowns. More generally, an attacker may target any CICO-k instance with $k \leq c$, so the permutation must be CICO-k secure for all $1 \leq k \leq c$ at level $\min\{q_2^k, 2^\kappa\}$.

- For the *compression mode*, a preimage attack fixes the χ output words to the target value but leaves all t input words free; a second-preimage attack additionally fixes the $t - \chi$ chaining-value inputs ($k_{\text{in}} = t - \chi$, $k_{\text{out}} = \chi$).[9] The permutation must resist these at level $\min\{q_2^\chi, 2^\kappa\} = 2^\kappa$ and $\min\{q_2^{\min(t-\chi,\chi)}, 2^\kappa\} = 2^\kappa$, respectively.

We do not explicitly consider algebraic *collision* attacks, since finding a collision requires either solving for two independent permutation evaluations simultaneously or directly modeling difference constraints. We claim that in both cases, the resulting systems are substantially larger than the considered systems, and that the generic birthday attack remains the most efficient strategy. An overview is given in Table 5.

Algebraic Models. We briefly introduce the considered algebraic models over F_m, F, and $\mathbb{F}_2$, based on the polynomial representations discussed in Section 5.2.2. Note that systems defined over a field smaller than the natural domain of the components must be augmented with field equations, to ensure zero-dimensionality of the corresponding ideal. In the following, the index $0 \leq r < \mathrm{R}$ runs over the rounds, with $r = -1$ reserved for the cipher input and c_{-1} the zero round constant by convention.

- *Over F_m.* We introduce $t - k_{\text{in}}$ input variables $X_{-1,i} \in F_m$ for the unconstrained input words, and $\mathrm{R}\,t$ S-box output variables $X_{r,j} \in F_m$ for $0 \leq r < \mathrm{R}$, $0 \leq j < t$. The S-box input at round r, word j, is the q_1-linearized polynomial L'_j from Equation (17) evaluated at the previous round's S-box outputs, plus the round constant. Hence, the S-box model $XY = 1$ yields one equation per S-box per round:

$$\left(\mathsf{L}'_j(X_{r-1,0}, \ldots, X_{r-1,t-1}) + c_{r-1,j}\right) \cdot X_{r,j} = 1\,, \tag{24}$$

  which is bilinear in $\{X_{r-1,i}\}$ and $X_{r,j}$ in the structural sense but has formal degree $q_1^{m-1} + 1$ due to the q_1-linearized linear layer. In total, the system over F_m consists of $\mathrm{R}\,t + k_{\text{out}}$ equations in $\mathrm{R}\,t + t - k_{\text{in}}$ unknowns: $\mathrm{R}\,t$ S-box equations of structural degree 2 (formal degree $q_1^{m-1} + 1$) and k_{out} output constraints of structural degree 1 (formal degree q_1^{m-1}). The k_{out} output constraints can be eliminated by q_1-linear Gaussian elimination, reducing the system to $\mathrm{R}\,t$ equations in $\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}}$ unknowns of structural degree 2.

[9] Note that due to the feed-forward $\mathsf{P}(x) + x$, the output constraints couple the permutation output with the input variables, so the compression preimage does not reduce to a pure CICO instance on P. However, we reuse the $k_{\text{in}}, k_{\text{out}}$ notation to facilitate presentation.

Table 6: Overview of the algebraic models over different fields with k_{in} constrained input words and k_{out} constrained output words over R rounds with state size t. Note that k_{in}, k_{out} and t are counted with respect to F_m, so variable and equation counts scale with the extension degrees $[F_m : F] = m$ and $[F_m : \mathbb{F}_2] = n_2$.

Field	#Equations	#Variables	Degree	Field eq.	Field eq. deg.
F_m	$\mathrm{R}\,t$	$\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}}$	$q_1^{m-1} + 1^{\dagger}$	no	-
F	$m\,\mathrm{R}\,t$	$m(\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}})$	2	yes	q_1
$\mathbb{F}_2$	$n_2\,\mathrm{R}\,t$	$n_2(\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}})$	2	yes	2

† Structural degree 2; the formal degree $q_1^{m-1} + 1$ arises from the q_1-linearized representation of the linear layer over F_m.

- *Over F.* We introduce $m(t - k_{\text{in}})$ input variables $x_{-1,i,k} \in F$ for the unconstrained input coordinates, and $m\,\mathrm{R}\,t$ S-box output variables $x_{r,i,k} \in F$ for $0 \le r < \mathrm{R}$. The k-th coordinate of the S-box input for word j at round r is a linear expression in the previous round's variables:

$$\mathsf{L}_k(x_{r-1,0,k}, \ldots, x_{r-1,t-1,k})_j + c_{r-1,j,k}\,, \tag{25}$$

  where $\mathsf{L}_k : F^t \to F^t$ denotes the linear map operating on the k-th row of the state matrix, as specified in Equation (8), and $(\cdot)_j$ selects the j-th output component. Using the polynomial representation of $XY = 1$ over F, cf. Equation (16), yields m bilinear equations per S-box per round, each of degree 2. In total, the system comprises $m(\mathrm{R}\,t + k_{\text{out}})$ equations in $m(\mathrm{R}\,t + t - k_{\text{in}})$ unknowns: $m\,\mathrm{R}\,t$ quadratic S-box equations and $m \cdot k_{\text{out}}$ linear output constraints. After substituting the linear output equations, this reduces to $m\,\mathrm{R}\,t$ quadratic equations in $m(\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}})$ unknowns. Each variable additionally satisfies the field equation $x^{q_1} - x = 0$ of degree q_1.

- *Over $\mathbb{F}_2$.* We introduce $n_2(t - k_{\text{in}})$ input variables and $n_2\,\mathrm{R}\,t$ S-box output variables over $\mathbb{F}_2$. As previously discussed, the linear layer becomes a $tn_2 \times tn_2$ binary matrix, and the m bilinear F-equations per S-box decompose into n_2 bilinear equations over $\mathbb{F}_2$, each of degree 2. After substituting the $n_2 \cdot k_{\text{out}}$ linear output constraints, the system reduces to $n_2\,\mathrm{R}\,t$ quadratic equations in $n_2(\mathrm{R}\,t + t - k_{\text{in}} - k_{\text{out}})$ unknowns. In contrast to the model over F, the field equations $u^2 + u = 0$ have the same degree as the S-box equations.

An overview of the resulting polynomial systems is given in Table 6. Prior to adding field equations, the system is square if and only if $k_{\text{in}} + k_{\text{out}} = t$, which holds for sponge (second) preimage with $t = m$ and for compression second preimage (both state widths). Otherwise ($k_{\text{in}} + k_{\text{out}} < t$), the system is underdetermined.

Attack Complexity and Round Number Derivation. We now derive the minimum number of rounds required for 128-bit security based on the following considerations regarding the complexity of Gröbner basis attacks in the three algebraic models. Detailed results are given in App. A.4.

- *Over F_m.* Current Gröbner basis implementations (e.g., Magma) do not, to the best of our knowledge, exploit the q_1-linearized structure of the linear layer, and no experiments beyond $m = 2$ terminated. To account for attackers potentially exploiting the structural degree 2, we report complexity estimates based on the Macaulay bound under the regularity assumption (cf. Table 14b). Since the bound depends only on R and t (the linear input/output constraints do not affect the nonlinear equation count), it is independent of the attack scenario. The estimated complexity exceeds 128-bit security at $\mathrm{R} = 5$ for $t = m$ and $\mathrm{R} = 3$ for $t = 3m/2$.

- *Over F and $\mathbb{F}_2$.* Both models yield degree-2 S-box equations but differ in variable count and field equation degree. Without field equations, the ideal is not zero-dimensional; we therefore add them to restrict solutions to the base field. Consequently, the solving degree is approximately the field equation degree of the respective model (cf. Table 14a). The two models trade variable count against field-equation degree and neither is uniformly easier; our experiments on small instances (Tables 15 and 16) confirm that the solving cost grows steeply with the number of rounds and, even more sharply, with the state size t and extension degree m. Already for these toy parameters the system leaves the feasible regime after only a few rounds, so the concrete instance ($m = 8$, $t \in \{8, 12\}$) is well beyond reach.

In summary, our observations indicate that $\mathrm{R} = 5$ rounds already make Gröbner basis attacks computationally infeasible.

Alternative Modeling Strategies. Recent works propose alternative modeling strategies that exploit the algebraic structure of inversion-based designs. Liu et al. [LMM24] leverage sparse linearized polynomials to derive additional dependencies, accelerating Gröbner basis attacks. Yang et al. [YZY25] show that linearized polynomials can be decomposed into lower-degree components and, by rewriting the S-box exponent and partially fixing the structure, the inversion map can be expressed as a linearized polynomial as well. However, these techniques are not effective in our setting, as they rely on few S-boxes, a small number of rounds, or sparse linearized representations of L.

Alternative Solving Strategies. The algebraic cryptanalysis of arithmetization-oriented hash functions has advanced rapidly, driven by the insight that solving the underlying polynomial system may be reduced (often at surprisingly low cost) to computing a single univariate polynomial in the ideal, whose roots reveal a solution. Two orthogonal directions realize this idea. The first stays within the Gröbner basis regime but chooses a dedicated monomial ordering under which the natural system is already, or almost, a Gröbner basis, so that the cost concentrates in the subsequent change of ordering [BBL+24, BBB+25a, CR25]. The second dispenses with Gröbner bases altogether, extracting the univariate polynomial by eliminating variables through the successive computation of resultants [YZY+24, BBB+25b], most recently using fast bivariate resultants [BBHN26]. Both approaches have so far only been demonstrated on strongly aligned SPNs over prime fields with an S-box that is a low-degree power map or has a low-degree inverse: Griffin, Arion, and Anemoi for the FreeLunch approach [BBL+24]; Poseidon, Neptune, and XHash8 for its CheapLunch extension [BBB+25a]; and *Rescue*, Griffin, Arion, and Anemoi for the resultant frameworks [YZY+24, BBB+25b, BBHN26].[10] While this does not by itself preclude their application to other structures, RainHash2.0 clearly departs from this template. In the following, we show that RainHash2.0 does not exhibit the structural properties required by these attacks, preventing their direct (efficient) application to our construction.

Both directions require the ideal to be zero-dimensional and, more restrictively, the polynomial system to be determined: the monomial-ordering approaches rest on a triangular, square system, and stepwise resultant elimination collapses to a single univariate polynomial only when the system is determined. The F and $\mathbb{F}_2$ models cannot satisfy those conditions simultaneously: without field equations their ideals are positive dimensional, whereas the field equations that restore zero-dimensionality render the system highly overdetermined. This leaves only the F_m model. For this model, however, the central premise of the monomial-order approaches fails structurally. In the S-box model the

[10] [YZY+24] also considered the inversion-based Jarvis [AD18], a block cipher attacked in a key-recovery rather than a CICO setting, and thus outside the scope considered here.

inversion relation reads $\mathsf{L}'_j(\dots) \cdot X_j = 1$, where L'_j is the q_1-linearized S-box input (see Equation (24)); it is bilinear in X_j and the earlier-round variables, so its leading monomial is a product of two distinct variables under any monomial ordering and can never be a pure power $X_j^{\alpha_j}$ required for the triangular Gröbner-basis structure underlying these attacks. The resultant frameworks avoid this structure but are limited instead by the degree growth during elimination, which their authors identify as the main bottleneck and counter with dedicated techniques: start-from-the-middle modeling, variable substitution, and fast interpolation in [YZY+24], and an efficient weighted-degree reduction based on fast multivariate multiplication in [BBB+25b]. However, these techniques remain effective only when the round functions have low degree. In the F_m model, the q_1-linearized layer gives each round relation a formal degree of $\approx q_1^{m-1} = 2^{56}$, placing the resulting polynomials far outside this regime. Finally, and independently of all the above, only CheapLunch is even within reach of our preimage attack scenarios (cf. Table 5): these fix $k_{\text{out}} = m/2 = 4$ output words, whereas every other work (and thus every resultant framework) is confined to CICO-1 and CICO-2.

5.3 Rebound Attacks

Rebound attacks [MRST09, LMR+09] are one of the most effective tools in the cryptanalysis of AES-like hash functions. Introduced by Mendel et al. at FSE 2009, they can be applied to both block cipher based and permutation based constructions. The idea of the rebound attack is to divide the attack into two phases, an inbound and an outbound phase:

- The aim of the inbound phase is to use the available degrees of freedom to find, at a low cost, a large number of pairs of values that satisfy a (part of a) differential path that would be very expensive to satisfy in a probabilistic way. In the case of AES or more generally of primitives with relatively small-size S-boxes (namely, 4, 6 or 8 bits S-boxes), the solutions for the inbound phase can be found by brute force. Otherwise, in order to efficiently find many solutions for this part of the characteristic, the attacker can solve a system of equations which describes the characteristic in this phase, as recently done in the case of Skyscraper [Bak25, BP25, BJGP25];
- In the outbound phase, each solution of the inbound phase is propagated outwards in both directions, while checking whether the characteristic also holds in this phase. The probability of the characteristic in the outbound phase should be as high as possible.

Assuming the probability of the outbound phase is ρ, then one needs to prepare $1/\rho$ starting points in the inbound phase to expect one pair conforming to the differential trail of the outbound phase. In the case of RainHash2.0:

- regarding the outbound phase, it can cover at most 2 rounds with probability 1, 3 with probability 2^{-56}, and 4 with probability 2^{-112}, based on the analysis given in Section 5.1.2;
- regarding the inbound phase, we do believe that the attacker cannot efficiently use brute-force to find solutions, due to the large size (namely, 64 bits) of the S-boxes. As discussed in Section 5.2.4, no more than 4 rounds can be covered using GB.

Still, it is crucial to keep in mind that using a truncated differential with low probability in the outbound phase impacts the maximum cost that the attacker can spend in the inbound phase, as

$$\text{cost(Inbound)} \times \text{cost(Outbound)} \leq 2^{128}\,.$$

Concretely, if the attacker aims to use the 3-round truncated differential with probability 2^{-56} (respectively, 4-round with probability 2^{-112}), then the cost of the inbound phase

must be smaller than 2^{72} (respectively, 2^{16}) for the overall attack to be successful. In these cases, the inbound phase can only cover strictly less than 4 rounds. Equivalently, if the attacker aims to cover 4 rounds in the inbound phase, then the outbound phase can only cover 2 rounds (as this is the longest known probability-1 truncated differential), for a total of 6 rounds. Putting all together, we claim that 8 rounds are sufficient to provide security against this attack.

6 Proof Implementation

In this section, we demonstrate the ZK-friendliness of RainHash2.0 by implementing proof circuits for two representative proving systems: Binius and VOLEitH-ZK. We first describe how the components of RainHash2.0 are mapped to each proving framework and then present the resulting practical evaluation.

6.1 RainHash2.0 for Binius

Here, we describe how RainHash2.0 can be proven within in the Binius PCS by detailing the requirements of the specific building blocks.

Substitution-Box. The S-box function $x \mapsto x^{-1}$ is represented in t committed columns. The correct result of the inversion function $y = x^{-1}$ is proven by the constraint $x \cdot y + 1 = 0$. To properly handle the special case $x = 0$, this constraint is extended with the additional term $x + \beta y = 0$, which ensures that $y = 0$ whenever $x = 0$. Here, β is a random challenge that prevents a dishonest prover from choosing $x, y \neq 0$ such that $x + y = 0$. The final constraint is given by $(x \cdot y + 1) \cdot (x + \beta y) = 0$, proving the correct computation of $y = x^{-1}$. For $x, y = 0$, the product evaluates to 0 because the right factor becomes $(0 + \beta \cdot 0) = 0$, while for any nonzero value $x = a$ and $y = a^{-1}$, the left factor becomes $(a \cdot a^{-1} + 1)$, which evaluates to 0, ensuring that the constraint $(x \cdot y + 1) \cdot (x + \beta y) = 0$ is satisfied when $y = x^{-1}$ or $y = 0$ if $x = 0$.

Linear Layer. For $t = 12$, the linear layer combines two state elements, selected via a right and left shift, with a constant field element α. This is captured using $8t$ virtual columns (one per row of the $8 \times t$ state matrix) via the virtual *linear combination* protocol. For $t = 8$, the linear layer reduces to a right shift, as $\alpha = 0$. While this still requires $8t$ virtual columns, constraint evaluation is simpler than for $t = 12$.

Field Size Switch. Within each round, the field size is switched twice: after the S-box layer, the state is decomposed as $\mathbb{T}_6^t \to \mathbb{T}_3^{8\times t}$, and after the linear layer, it is recomposed into $\mathbb{T}_6^t$. The composition of t elements in $\mathbb{T}_6$ can be expressed as a linear combination of the $8t$ values in $\mathbb{T}_3$ and their respective shift offsets: $\sum_{i=0}^{7} x_i \cdot 2^{3(7-i)}$. Proving correct decomposition proceeds as follows: the decomposed state is committed in $8t$ $\mathbb{T}_3$-columns, the composition is recomputed to obtain t values in $\mathbb{T}_6$, and equality constraints enforce that each recomposed value matches the original $\mathbb{T}_6$ element.

Total. In summary, one round function requires:

- t $\mathbb{T}_6$ and mt $\mathbb{T}_3$ committed columns, representing the state after the S-box computation and the decomposed state;
- $(8 + 2)t$ virtual columns, representing the state after the shuffle layer and the recomposed states before and after the shuffle;

- $2t$ $\mathbb{T}_6$ zero-constraints, proving the correctness of the S-box computation and the state decomposition;
- $2t$ $\mathbb{T}_6$ and $8t$ $\mathbb{T}_3$ virtual constraints, proving the correctness of the shuffle and state composition.

Note that for $t = 12$, not only the number of constraints but also the complexity of the shuffle layer increases.

6.2 RainHash2.0 for VOLE-ZK

The VOLE-ZK proof system consumes the secret input $x \in \mathbb{F}_2^{64 \cdot t}$ as part of the witness. The proof proceeds round by round as follows.

State Representation and the Linear Layer L. Before the first round, the state is transformed into row-major representation and packed into $\mathbb{F}_{2^8}$ elements as $\mathbb{F}_2^{64 \cdot t} \to \mathbb{F}_{2^8}^{8 \cdot t}$. The L layer is a linear VOLE operation, involving indexing switching, on $\mathbb{F}_{2^8}$ elements, applied row-wise. In the proof implementation, we operate directly on $\mathbb{F}_2$ bit indices rather than interpreting them as $\mathbb{F}_{2^8}$ elements. This avoids repeated representation switches between the row-major (L) and column-major (S) layouts per round, which would otherwise incur cache miss penalties and reduce runtime performance.

The Nonlinear Layer S and Inverse Checks. The S layer is the inverse S-box over $\mathbb{F}_{2^{64}}$, defined as $\mathsf{S}(x) = x^{-1}$ for $x \neq 0$ and $\mathsf{S}(0) = 0$. Before the VOLE inverse check, the $\mathbb{F}_{2^{64}}$ witness elements in the circuit are lifted to $\mathbb{F}_{2^\lambda}$ to ensure correct ZK soundness with respect to the security parameter λ; these lift gates are linear in VOLE and require no additional communication. For $x \neq 0$, the inverse check follows the *Quicksilver* strategy [YSWW21] as adapted in VOLEitH FAEST signature [BBdSG+23, BBM+24]: ${}^rW_{\text{in}}^i \cdot {}^rW_{\text{out}}^i + 1 \stackrel{?}{=} 0$, where ${}^rW_{\text{in}}^i$ and ${}^rW_{\text{out}}^i$ denote the input and output of the i-th S-box in round $r \in \{0, \ldots, 7\}$, respectively. To also handle $x = 0$, [BBM+24] propose replacing the above with two checks:

$$\left({}^rW_{\text{in}}^i\right)^2 \cdot {}^rW_{\text{out}}^i + {}^rW_{\text{in}}^i \stackrel{?}{=} 0, \tag{26}$$

$${}^rW_{\text{in}}^i \cdot \left({}^rW_{\text{out}}^i\right)^2 + {}^rW_{\text{out}}^i \stackrel{?}{=} 0. \tag{27}$$

These incur no additional communication overhead, as squaring is linear over $\mathbb{F}_2$. RainHash2.0 with $t = 8$ (resp. $t = 12$) requires 8 (resp. 12) inverse checks per round.

Round Transition and Final Check. After each S layer, the outputs ${}^rW_{\text{out}}^i$ (in $\mathbb{F}_2$) are passed through L followed by a round-constant addition $\oplus\, c_r$, producing the input to the next round's S-boxes. Finally, in the last round, the public output $y \in \mathbb{F}_2^{2\lambda}$ and the hash capacity (secret witness) are combined into $\mathbb{F}_2^{64 \cdot t}$ and processed through RainHash2.0's inverse permutation (the linear round-constant addition $\oplus\, c_7$ followed by the inverse shift-rows) to recover the last-round S-box outputs ${}^7W_{\text{out}}^i$. The final VOLE inverse check is then ${}^7W_{\text{in}}^i \cdot {}^7W_{\text{out}}^i + 1 \stackrel{?}{=} 0$.

6.3 Practical Results

We provide proof implementations of RainHash2.0 with $t \in \{8, 12\}$.

6.3.1 Binius Circuits

The Binius implementation constructs a proof attesting to the correct execution of concatenated RainHash2.0 permutations, thereby demonstrating knowledge of a valid hash preimage. By benchmarking 2^{14} permutations, we evaluate the cost of proving a RainHash2.0 hash over a 1 MB message. Our constraint system for RainHash2.0 is incorporated into the commitment phase and linked with the existing proving and verification functions, resulting in an end-to-end proof implementation. We compare our proof construction against existing proof implementations[11], evaluating performance based on proof size, proving time, and verification time, depending on the hash output size. All benchmarks were conducted on an AMD Ryzen 9 7900X (12 cores), with Rust code compiled in release mode using the `-C target-cpu=native` flag; results are summarized in Table 7.

Table 7: Benchmark results for the proof implementations in Binius when hashing ≈ 1 MB† of data, supporting single- and multi-threaded prover and verifier.

Permutation	n	t	#Perm.	Proof size (KiB)	Proving time (s)		Verify time (ms)	
					single	*multi*	*single*	*multi*
Keccak-f [BDPV13]	64	24	2^{13}	438	3.038	0.425	45.70	46.59
SHA-256 [Nat15]	32	16	2^{14}	701	5.248	1.383	233.55	235.30
Grøstl-P [GKM+09]	8	64	2^{14}	416	0.495	0.170	114.97	115.81
Vision Mark-32 [AMPŠ24]	32	24	2^{14}	560	1.764	0.605	10.12	10.60
Anemoi [BBC+23]	32	16	2^{14}	479	1.497	0.499	12.28	9.18
	32	24	2^{14}	581	2.491	0.885	12.05	12.31
	64	8	2^{14}	510	2.364	0.708	5.78	6.25
	64	12	2^{14}	528	2.832	0.854	8.43	7.04
	128	4	2^{14}	491	2.826	0.712	4.53	4.74
	128	6	2^{14}	502	3.522	0.895	5.15	5.14
Poseidon2b [GKK+26]	32	16	2^{14}	402	0.370	0.129	4.66	4.50
	32	24	2^{14}	471	0.469	0.175	5.24	5.45
	64	8	2^{14}	366	0.477	0.143	3.61	3.75
	64	12	2^{14}	426	0.748	0.207	4.10	4.34
	128	4	2^{14}	386	1.255	0.304	3.54	3.66
	128	6	2^{14}	400	1.432	0.347	3.73	3.84
Poseidonb [GKK+26]	32	16	2^{14}	411	0.434	0.150	4.53	4.84
	32	24	2^{14}	519	0.692	0.230	5.67	5.96
	64	8	2^{14}	370	0.535	0.162	3.68	3.82
	64	12	2^{14}	432	0.837	0.236	4.92	4.49
	128	4	2^{14}	386	1.255	0.302	3.56	3.67
	128	6	2^{14}	400	1.437	0.343	3.73	3.83
RainHash2.0	64	8	2^{14}	425	0.649	0.175	9.71	10.31
	64	12	2^{14}	464	0.839	0.231	13.23	13.77

† For Keccak-f, we prove 2^{13} permutation calls, which results in a hash size of 1.1 MB. We get a hash output size of 1.049 MB for all other permutation functions for 2^{14} permutation invocations.

Keccak-f exhibits significantly higher proving and verification times than RainHash2.0, with comparable proof sizes, while SHA-256 performs even worse across all metrics. Grøstl-P achieves slightly lower single-threaded proving times; in the multi-threaded setting it matches the 512-bit RainHash2.0 variant, while the 768-bit variant remains slightly slower.

[11] The Binius framework already provides proof implementations for Keccak-f [BDPV13, BDPA11b], SHA-256 [Nat15], Grøstl-P [GKM+09], and Vision Mark-32 [AMPŠ24]. See https://gitlab.com/IrreducibleOSS/binius/-/tree/924103fe50a9767b03c61981a5df12dafc1f44bd. Proof implementations of Anemoi [BBC+23] and Poseidon2b [GKK+26] are provided by [GKK+26] and taken from the corresponding Github repository https://github.com/Poseidon-Hash/Poseidon2b.

Table 8: VOLE-ZK circuit benchmarks of different hash functions. s and f denote *small* and *fast* VOLEitH ZK versions respectively: s achieves smaller proof sizes at the cost of higher prover/verifier runtime, while f makes the opposite trade-off.

Scheme (s/f)	Proof size (KB)	Proving time (ms)	Verify time (ms)
†SHAKE256-deg2$_s$	55.5	31.89	33.37
†SHAKE256-deg2$_f$	80.1	5.57	6.03
†SHAKE256-deg16$_s$	14.9	25.88	11.61
†SHAKE256-deg16$_f$	21.1	17.87	4.74
†RainHash-512$_s$	6.2	15.33	14.54
†RainHash-512$_f$	10.03	2.72	2.67
RainHash2.0-512$_s$	6.8	12.64	12.16
RainHash2.0-512$_f$	11.03	0.42	0.41
RainHash2.0-768$_s$	9.9	26.45	21.2
RainHash2.0-768$_f$	16.53	7.82	7.65

† We benchmark the circuits from [BBB+26] with our VOLE-ZK parameters for consistent comparison.

However, its verification times are significantly higher, and proof sizes slightly smaller. Vision Mark-32 requires roughly twice the proving time of RainHash2.0, with verification times between the 512-bit and 768-bit RainHash2.0 variants and moderately larger proofs. Anemoi exhibits substantially higher proving times (at least 2×), while its verification times range from comparable ($n = 32$) to significantly lower ($n \in 64, 128$); proof sizes are slightly larger. For Poseidon(2)b, $n = 32$ yields up to 2× faster proving, $n = 64$ is comparable to RainHash2.0, and $n = 128$ is up to 2× slower than RainHash2.0. Verification times are consistently 2-3× lower than RainHash2.0, while proof sizes are slightly larger. Overall, only Anemoi and Poseidon(2)b achieve competitive performance, combining lower verification times with comparable proving times. In contrast, hardware-oriented designs such as SHA-256 and Keccak-f are clearly outperformed by RainHash2.0 across all considered metrics.

6.3.2 VOLE Circuits and Plain AVX

The RainHash2.0 VOLE proof is implemented in the FAESTv2.0 [BBB+25c, BBB+25d] framework using VOLEitH-ZK and its optimizations. The framework provides two configurations: small s and fast f, trading proof size for proving/verification time, determined by the size and number of the Merkle trees in the all-but-one vector commitment openings (for soundness w.r.t. λ). While FAEST defaults to $s = 11$ (for AES), we use $s = 9$ for RainHash2.0 to reduce proof size with negligible runtime impact. For meaningful comparison, we also set $s = 9$ for RainHash and SHAKE256 circuits from [BBB+26], and port their implementation from FAESTv1.0 to FAESTv2.0 to benefit from recent optimizations.[12] Results are summarized in Table 8. We observe that RainHash achieves the smallest proof size due to one fewer round than RainHash2.0-512. Nevertheless, RainHash2.0, in particular RainHash2.0-512, achieves competitive small proof sizes compared to SHAKE256, while offering advantages in plain and hardware performance. Using the inverse-norm optimization [BBB+25d] for the S-box would further reduce the witness size by 25%. We conservatively estimate RainHash2.0-512$_s$ and RainHash$_s$ proofs to be close at around 6 KB, and RainHash2.0-768$_s$ at around 9 KB.

As indicated in Section 2.4, plain performance is strongly affected by large-field inversions. While RainHash uses full-state inverses, leading to quadratic-cost field arithmetic (e.g., via `clmul`), RainHash2.0 uses several smaller $\mathbb{F}_{2^{64}}$ S-boxes per round, yielding a

[12] To the best of our knowledge, SHAKE256, RainHash, and RainHash2.0 are currently the only hash functions implemented in VOLE-ZK over $\mathbb{F}_{2^k}$.

Table 9: 1 MB message hashing runtime of optimized plain implementation.

Permutation	Plain runtime (μs)
SHA3-256 [Dwo15]	180
GMiMC [AGP+19]	52 616
Rescue-Prime [SAD20]	496 460
Poseidon [GKR+21]	26 375
RainHash-512 [BBB+26]	230 548
RainHash2.0-512	36 344
RainHash2.0-768	28 209

$\approx 15\times$ speed-up. Moreover, replacing the MDS matrix in RainHash with the new shift-rows-like approach in RainHash2.0 improves plain performance by $27\times$, as $\mathbb{F}_2$ matrix-vector multiplications are significantly slower than $\mathbb{F}_{2^8}$ shift operations. Overall, RainHash2.0 achieves substantial gains in plain performance – up to $8\times$ faster than RainHash – and remains competitive with the industry-standard Poseidon in the 768-bit setting (Table 9).

7 Hardware Implementation

Hardware acceleration is a central design aspect of ZK primitives. Therefore, we present an FPGA-based implementation and hardware results of RainHash2.0 in this section. In addition, we implement related hash functions on hardware and compare them to our RainHash2.0 results. For all experiments, we use Xilinx Vivado and Vitis 2022.2 with performance settings for synthesis, place, and route. We target an Alveo U280 datacenter FPGA [Xil23], integrate our designs into an XDMA/PCI streaming shell, and verify their correctness by running all experiments on the real FPGA.

7.1 Hardware Design for RainHash2.0

We optimize our RainHash2.0 hardware implementation for high throughput and high clock frequency. Therefore, we fully unroll one RainHash2.0 permutation $\mathsf{P}_{64,t}$ by instantiating 8 pipelined round function modules, each computing one round $\mathsf{R}^{(r)}$ with hardcoded round constants. Our permutation design operates at a high clock frequency of 500 MHz and accepts one new input in every clock cycle. This throughput-oriented design strategy with fully unrolled and pipelined round functions suits the data-centric ZK deployment and aligns with existing works [AMPŠ24, HKPR25]. The next subsections detail our hardware design of RainHash2.0's nonlinear and linear layers. We use the tower field construction explained in Section 2.1 for the low-level arithmetic and present optimizations to enhance the overall hardware efficiency.

7.1.1 Hardware Implementation of the Non-Linear Layer S

The nonlinear layer S is the most resource-intensive operation within the RainHash2.0 round function. Therein, the $\mathbb{T}_3^{8\times t}$ state matrix is interpreted as a $\mathbb{T}_6^t$ vector, and the vector elements are inverted in t parallel $\mathbb{T}_6$-field inversion modules. Our inversion modules (named Inv_k for $\mathbb{T}_k$ inversion) follow the recursive design shown in Figure 6. The Inv_k module instantiates addition, squaring, constant multiplication, inversion, and multiplication sub-modules. On the top level, this recursive construction starts with $k = 6$ to compute the inversion of one $\mathbb{T}_6$ field element. Subsequently, the sub-modules of Inv_k operate over $\mathbb{T}_{k-1}$ via a divide-and-conquer approach. Each submodule (except the trivial addition using Xor) also uses a recursive decomposition similar to Karatsuba's approach (see Section 2.1).

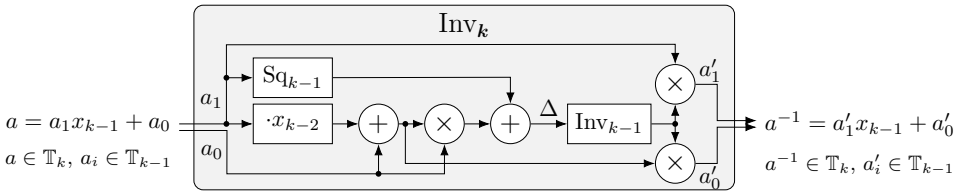


Figure 6: Hardware design for $\mathbb{T}_k$ tower field inversion. Refer to Section 2.1 for details.

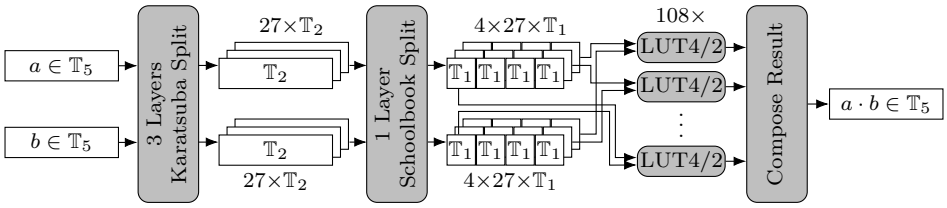


Figure 7: Multiplication architecture for $\mathbb{T}_5$. We split each operand (a, b) into 4×27 elements in $\mathbb{T}_1$ via the Karatsuba and schoolbook methods. Then, 108 $\mathbb{T}_1$-multiplications are performed, each using one 4-input/2-output LUT.

Multiplier Architecture and Optimal Recursion Depth. Within the recursive Inv$_k$ module, the three $\mathbb{T}_{k-1}$ multiplication modules dominate the area consumption. We hence optimize the recursive multiplier architecture relying on the Karatsuba decomposition and empirically search for the optimal recursion depth. This reveals that the recursive decomposition to the bit-level with $k = 0$ (i.e., $\mathbb{T}_0 \cong \mathbb{F}_2$) does not yield optimal area results on FPGAs due to their fundamental lookup-table (LUT) architecture [KSK25]. Specifically, modern Xilinx FPGAs [AMD25] support LUTs with up to 6 inputs and 2 outputs, which allows us to stop the recursive Karatsuba decomposition when reaching $\mathbb{T}_2 \cong \mathbb{F}_{16}$. At this stage, we use one layer of $\mathbb{T}_2$-schoolbook decomposition followed by LUT-based $\mathbb{T}_1$ multiplication, as shown in Figure 7. The operands in $\mathbb{T}_1 \cong \mathbb{F}_4$ multiplication are 2-bit wide, which allows efficient computation of the 2-bit result using one FPGA-native 4-input/2-output LUT (LUT4/2 in Figure 7). Therein, the four inputs to each LUT4/2 are the two operands from $\mathbb{T}_1$, and each LUT4/2 output readily computes one bit of the result.

We also apply this technique to the other sub-modules and verify the effectiveness via Xilinx Vivado 2022.2. Therein, we observe a LUT reduction for $\mathbb{T}_6$-inversion from 3,139 LUTs to just 1,976 LUTs, offering a clear area-related benefit for hardware designs.

7.1.2 Hardware Implementation of the Linear Layer L

Unlike the S-layer, RainHash2.0's linear layer L, presented in Section 3.1, is lightweight. In our fully unrolled FPGA design, the rotations and bit-wise round constant additions do not consume logic resources, since constant rotations are free in hardware and the round constant additions are merged into subsequent LUTs. The only operation in L consuming logic resources is the constant multiplication by α and XOR-ing its result for the $t = 12$ variant of RainHash2.0 (see Equation (8)). In our implementation, we use $\alpha = x_2 \in \mathbb{T}_3$ for $t = 12$ and $\alpha = 0$ for $t = 8$.

7.1.3 FPGA Integration and Data Movement

To run our RainHash2.0 permutation on the real Alveo U280 FPGA, we integrate the permutation module into a PCI shell based on the Xilinx XDMA IP [Xil22]. The PCI shell

drives the FPGA's PCIe Gen3x16 interface and DDR4 memories, and performs simple FIFO-based data buffering. We assume the host CPU to perform all data alignment and packing operations such that no reordering functionality is left to the hardware.

Our fully unrolled and pipelined RainHash2.0 permutation unit maximizes hashing throughput of data-independent permutation evaluations, where the throughput (TP) for a sponge construction (capacity $c = 256$ bits, one fully unrolled permutation unit) is computed by $\text{TP} = (t \cdot 64 - c) \cdot 500\ \text{MHz} \cdot 10^{-9}$. For RainHash2.0 with $t = 8$, the hashing throughput is TP=128 Gbps. In this case, we stream the hashing data from the host CPU via PCI to the permutation unit. The used PCI interface offers up to 126 Gbps, making this configuration slightly I/O-bound. Considering RainHash2.0 with $t = 12$, the throughput is TP=256 Gbps. This is too high for the PCI interface and would result in a highly I/O-bound design. Thus, we assume the hashing data in the FPGA's DDR4 memories as in [HKPR25]. We stream the data from the DDR4 to the permutation unit with an aggregate bandwidth of 307 Gbps, making the design compute-bound.

In case of lower off-chip bandwidths, which cannot supply the hashing TP of a fully unrolled permutation, we propose to instantiate fewer round function modules and iterate the round function modules multiple times to compute a full permutation. For example, only instantiating 4 (instead of 8) round function modules requires two passes through the pipeline. This halves the throughput and approximately halves the area consumption. Experiments show a negligible area overhead of less than 2% due to additional multiplexing within the pipeline and for switching round constants between the passes. Thus, the *throughput-per-area-ratio* of the fully unrolled and the iterated design is almost identical, thereby offering an area-performance tradeoff without significantly affecting the overall hardware efficiency.

Permutation Latency. Our RainHash2.0 hardware design optimizes for high throughput, which aligns with data-centric server-side deployment of ZK hardware accelerators. Therefore, a single RainHash2.0 round function module features 27 pipeline stages, and the whole permutation module with 8 round function modules has $8 \cdot 27 + 3 = 219$ pipeline stages; the initial linear layer, input, and output buffering each add one pipeline stage.

This high pipeline latency is acceptable in typical applications of hardware acceleration, where many independent operations demand throughput-oriented hashing. Yet, some applications have inter-permutation dependencies, which prevent simultaneous permutation processing in a deep pipeline. To mitigate these pipeline hazards, the pipeline can be flushed, which causes throughput reductions proportional to the pipeline latency. However, as shown in Table 11, our RainHash2.0 hardware design has a lower latency than other AO hash functions, reducing the impact on throughput in data-dependent workloads.

7.1.4 Area and Power Consumption of RainHash2.0 in FPGA

Table 10 presents the area consumption of our fully unrolled and pipelined RainHash2.0 permutation modules for $t = 8$ and $t = 12$. The targeted Alveo U280 FPGA offers a total of 1,131k lookup tables (LUTs) and 2,265k single-bit flip-flops/registers (REGs). We use an Xilinx XDMA IP [Xil22] to establish a PCI Express interface for streaming data between the host CPU and the FPGA accelerator running our RainHash2.0 instance. This XDMA interface consumes about 60k LUTs, 61k REGs, and 172 BRAMs.

Both RainHash2.0 versions have 8 rounds, and one round function module consumes 15.8k and 24.6k LUTs as well as 12.4k and 19.9k registers for $t = 8$ and $t = 12$, respectively. Within one round, the nonlinear S layer with the Inv_6 modules clearly causes the major area consumption, while the linear L layer only contributes negligibly to the LUT and register consumption. Neither of the two RainHash2.0 permutations use DSPs or BRAMs. Furthermore, the $t = 8$ permutation is about $1.6\times$ smaller than the $t = 12$ permutation. This matches the expectations given the $1.5\times$ smaller state size of RainHash2.0 with $t = 8$.

Table 10: Post-implementation area consumption of fully unrolled and pipelined RainHash2.0 permutations $\mathsf{P}_{64,t}$ with 8 rounds. The area results vary slightly across instances of the same module due to Vivado's optimizations. No DSPs are used.

Module Hierarchy	Instances	LUTs		REGs		BRAMs	
Alveo U280 FPGA Total	1	1,131k	100%	2,265k	100%	2k	100%
PCI Express Shell with XDMA	1	59,647	5.3%	60,792	2.7%	172	8.5%
RainHash2.0 Permutation $\mathsf{P}_{64,8}$	1	126,586	11.2%	100,577	4.4%	0	0.0%
|- Single Round $\mathsf{R}^{(r)}$	8	15,753	1.4%	12,372	0.5%	0	0.0%
|- L Layer	1	0	0.0%	2,049	0.1%	0	0.0%
|- S Layer	1	15,753	1.4%	10,323	0.5%	0	0.0%
|- Inv_6	8	1,976	0.2%	1,290	0.1%	0	0.0%
|- Mul_5	3	438	0.0%	292	0.0%	0	0.0%
|- Inv_5	1	663	0.1%	414	0.0%	0	0.0%
RainHash2.0 Permutation $\mathsf{P}_{64,12}$	1	197,798	17.5%	162,871	7.2%	0	0.0%
|- Single Round $\mathsf{R}^{(r)}$	8	24,625	2.2%	19,904	0.9%	0	0.0%
|- L Layer	1	542	0.0%	1,344	0.1%	0	0.0%
|- S Layer	1	24,083	2.1%	15,483	0.7%	0	0.0%
|- Inv_6	12	1,976	0.2%	1,290	0.1%	0	0.0%

Finally, we note a higher LUT consumption for the Inv_6 module in Table 10 (1,976 LUTs) compared to the initial results in Table 3 (1,399 LUTs). This is explained by the pipelining of the final design (Table 10), while the initial results from Table 3 do not include the overhead of pipelining. Although pipelining slightly increases the area consumption, it is highly effective at improving the throughput and the efficiency of the hardware design.

Considering the post-implementation power consumption, our RainHash2.0 permutation unit consumes 24.4 W and 37.5 W for $t = 8$ and $t = 12$, respectively. Since the PCI Shell and XDMA additionally require 8.0 W, the overall system power consumption is 32.4 W and 45.5 W for $t = 8$ and $t = 12$, respectively.

7.2 Comparison to Related Works

After presenting the details of our RainHash2.0 hardware implementation, we now compare the hardware efficiency of RainHash2.0 to related hash functions. The comparison includes general-purpose hash functions (SHA-2 [Gam23], SHA-3 [HKPR25], and TurboSHAKE [BDH+23]) and arithmetization-oriented hash functions over binary fields (Anemoi [BBC+23], Poseidon(2)b [GKK+26], Vision Mark-32 [AMPŠ24], and RainHash [DKR+22]). We develop FPGA implementations for all arithmetization-oriented hash functions lacking publicly available FPGA designs. Thereby, we ensure a fair comparison by using the same design strategies, finite-field arithmetic modules, and matching optimizations, as presented in Section 7.1.

The comparison presented in Table 11 mainly focuses on the throughput-per-LUT (kbps/LUT) efficiency benchmark for sponge constructions defined in [AMPŠ24], where higher values indicate more efficient designs. As shown, the $t = 12$ version of RainHash2.0 consumes 198k LUTs and runs at a clock frequency of 500 MHz, resulting in a hashing throughput of TP = 256 Gbps and an efficiency benchmark of 1,294 kbps/LUT. Compared to the $t = 12$ version, the $t = 8$ version of RainHash2.0 has a smaller state size and therefore consumes 1.56× fewer LUTs. Yet, RainHash2.0 with $t = 8$ reaches a 1.28× lower efficiency of 1,011 kbps/LUT, due to the lower rate/capacity ratio of the sponge construction. Table 11 also shows RainHash2.0's single-permutation latency of 438 nanoseconds (ns).

In the next paragraphs, we provide a detailed comparison of RainHash2.0 to the related designs. We limit our discussion to the most efficient configuration of RainHash2.0 for hardware ($t = 12$), while providing the remaining benchmarks in Table 11.

Table 11: FPGA performance comparison considering one fully unrolled permutation.

Hash Function, S-box Size	AO	Rate Bits	Freq. MHz	LUT k	Latency ns	TP Gbps	Efficiency kbps/LUT	Our Efficiency Benefit
RainHash2.0, 64b, $t = 8$[a]	✓	256	500	127	438	128	1,011	**1.28**×
RainHash2.0, 64b, $t = 12$[a]	✓	512	500	198	438	256	1,294	**1.00**×
Vision Mark-32, 32b [AMPŠ24][b]	✓	512	250	398	448	128	322	**4.02**×
Anemoi, 63b [BBC+23][a]	✓	252	465	556	1,974	117	211	**6.14**×
Rain, 128b [DKR+22][a]	✓	128	500	230	962	64	278	**4.66**×
Rain, 256b [DKR+22][a]	✓	256	429	748	1,569	110	147	**8.82**×
Poseidon2b, 32b [GKK+26][a]	✓	256	500	220	760	128	583	**2.22**×
Poseidon2b, 32b [GKK+26][a]	✓	512	499	330	762	255	773	**1.68**×
Poseidon2b, 64b [GKK+26][a]	✓	256	484	467	1,380	124	265	**4.88**×
Poseidon2b, 64b [GKK+26][a]	✓	512	443	653	1,508	227	347	**3.73**×
PoseidonB, 32b [GKK+26][a]	✓	256	467	451	749	120	265	**4.88**×
PoseidonB, 32b [GKK+26][a]	✓	512	446	881	785	228	259	**5.00**×
PoseidonB, 64b [GKK+26][a]	✓	256	416	728	1,524	106	146	**8.85**×
SHA-2, 32b [Gam23][1,2,c]	×	256	274	2	124	4	2,400	**0.54**×
SHA-3, 5b [HKPR25][a]	×	1,088	500	69	52	544	7,870	**0.16**×
TurboSHAKE, 5b [BDH+23][a]	×	1,088	500	26	28	544	20,852	**0.06**×

AO: arithmetization-oriented, [a]Alveo U280, [b]Alveo U55C, [c]Kintex xcku035-ffva1156-3-e, [1]partially unrolled, [2]Merkle-Damgård construction. No DSPs or BRAMs are used in any of these designs.

Vision Mark-32. We begin with Vision Mark-32 [AMPŠ24]. Similar to RainHash2.0, Vision Mark-32 operates over binary tower fields. The authors also present hardware results for an Alveo U55C FPGA, which is very similar to our Alveo U280 FPGA, thereby allowing a direct comparison. Yet, Vision Mark-32 only reaches a frequency of 250 MHz, which limits its hashing throughput. Moreover, Vision Mark-32 uses 398k LUTs due to costly MDS matrix multiplications in the linear layer. In contrast to Vision Mark-32, RainHash2.0 with $t = 12$ uses a simpler linear layer and lowers the area consumption by 2× for the same rate of $r = 512$. This allows RainHash2.0 to reach a 4.02× efficiency improvement over Vision Mark-32.

Anemoi. Next, we consider the Anemoi [BBC+23] permutation with its SPN-based round function. Instantiated over $\mathbb{F}_{2^{63}}$ and with a state size of $2 \times 4 \times 63 = 506$ bits, Anemoi consumes 556k LUTs. This high LUT consumption is caused by the $x \mapsto x^{1/3}$ operation in the Anemoi S-box. We implement this operation using $x^{1/3} = x^{3^{-1} \bmod 2^{63}-1} = x^{0x5555555555555555}$ in $\mathbb{F}_{2^{63}}$ and via an efficient addition chain obtained from [McL25]. Overall, the computation of $x^{1/3}$ requires 5 multiplications and 5 squarings. Together with other multiplications in the S-box and the linear layer, this leads to the high area consumption of Anemoi and also impacts the achievable clock frequency of the hardware design. In particular, the dense design reaches only 465 MHz due to timing closure and congestion issues. These effects limit the hardware efficiency of Anemoi, which is up to 6.14× less efficient than RainHash2.0.

Rain. The Rain design [DKR+22] proposes a keyed permutation over large fields $\mathbb{F}_{2^n}$ with $n \in \{128, 192, 256\}$. The S-box in Rain causes the major area consumption, as it computes $x^{-1} = x^{2^n-2}$ in $\mathbb{F}_{2^n}$. Similar to Anemoi, we implement the S-box using an addition chain with 10 multiplications for the $n = 128$-bit and $n = 256$-bit cases. Table 11 shows the resulting hardware benchmarks of the Rain permutation. We note that Rain with $n = 256$ only reaches 429 MHz, which is explained by the routing congestion in the large 256-bit multiplications. As a result, Rain is 4.66× ($n = 128$) and 8.82× ($n = 256$) less efficient than RainHash2.0.

Poseidon(2)b. The authors of [GKK+26] propose PoseidonB and Poseidon2b permutations. Both permutations use $x \mapsto x^7$ power maps for the S-box and split the permutation

into outer and inner rounds. The difference between PoseidonB and Poseidon2B lies in the linear layer of the outer rounds; PoseidonB uses MDS matrices in the outer rounds, while Poseidon2B uses circular matrices with low Hamming-weight elements. The heavy-weight MDS matrix multiplications in PoseidonB result in increased area consumption compared to Poseidon2B, as shown in Table 11. Indeed, the PoseidonB variant with $\mathbb{F}_{2^{64}}$ and 768-bit state even exceeds the available LUTs in the target FPGA. Compared to Poseidon(2)B, our RainHash2.0 reaches between 1.68× and 8.85× higher efficiency.

General-purpose hash functions. Finally, we consider conventional SHA-2, SHA-3, and TurboSHAKE hash functions. The hardware implementation of SHA-2-256 from [Gam23] features a partially unrolled SHA-2 core running at 274 MHz. Therefore, the efficiency benchmark is 2,400 kbps/LUT, which is about 1.85× more efficient than the best-performing configuration of RainHash2.0. The native performance of SHA-3 and TurboSHAKE (a round-reduced version of the SHA-3 permutation) is even higher. Compared to RainHash2.0, SHA-3 and TurboSHAKE run at 500 MHz and reach a 6.1× and 16.1× higher efficiency, respectively.

7.3 Discussion of Hardware Results

Based on the presented hardware results, we see a clear efficiency benefit of RainHash2.0 compared to other arithmetization-oriented hash functions over binary fields. Specifically, RainHash2.0 outperforms other circuit-friendly designs by factors between 1.68× and 8.85×. Compared to conventional hash functions, the performance of RainHash2.0 is naturally lower. However, conventional hash functions introduce overheads in their arithmetization, leading to higher prover and verifier costs.

Moreover, the RainHash2.0 hardware benefits from a compact area footprint and improved routability. These properties, combined with hardware-friendly binary field arithmetic, enable our RainHash2.0 implementation to reach a high clock frequency of 500 MHz, which directly translates to high hashing throughput. Reaching such high frequencies on FPGAs is not trivial and shows the strength of our proposed design. Finally, our RainHash2.0 design strategy for FPGAs naturally extends to other hardware platforms such as ASICs, thereby allowing broad-scale hardware deployment of RainHash2.0.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions, which helped improve this paper.

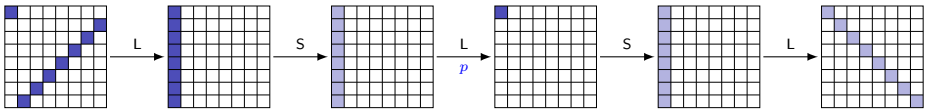
Sujoy Sinha Roy was partially supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

Lorenzo Grassi was supported by the European Research Council (ERC), grant number 101160608 “SYMPZON”. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

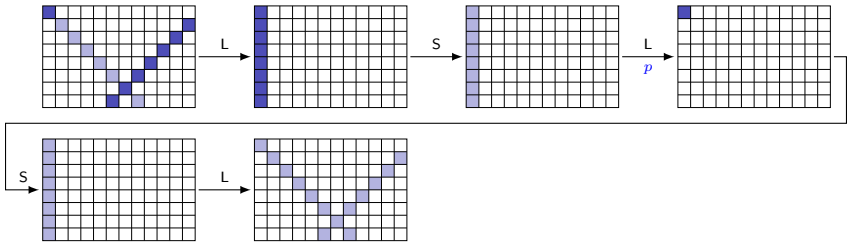
Supporting Material

A Supplementary Figures and Tables

A.1 Statistical Attacks - Truncated Differentials

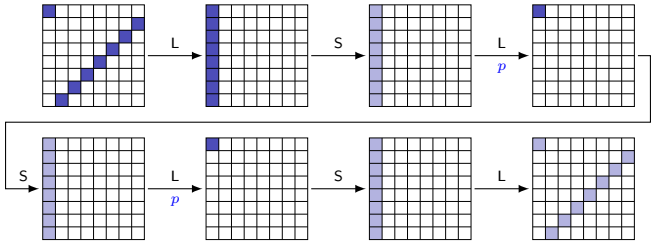


(a) $t = 8$ with $\alpha = 0$.

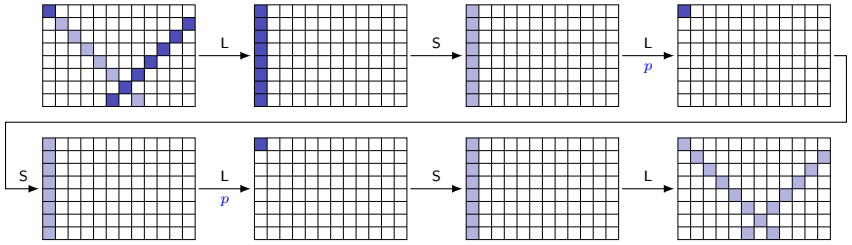


(b) $t = 12$ with $\alpha \neq 0$.

Figure 8: Truncated differential for 2-round RainHash2.0. A byte with non-zero difference is denoted in blue. The probability p is equal to 2^{-56}.



(a) $t = 8$ with $\alpha = 0$.



(b) $t = 12$ with $\alpha \neq 0$.

Figure 9: Truncated differential for 3-round RainHash2.0. A byte with non-zero difference is denoted in blue. The probability p is equal to 2^{-56}.

A.2 Algebraic Attacks - Polynomial Representations

Table 12: Polynomial representations of the inversion map $\mathsf{S}_j : \mathbb{T}_{k_2} \to \mathbb{T}_{k_2}$ in coordinates over $\mathbb{T}_{k_1}$, cf. Equations (13) and (16). For each pair (k_2, k_1), the table lists the extension degree $m = 2^{k_2-k_1}$, the monomial bound as specified in Table 4, as well as the maximum/minimum/average number of monomials in the different models. All m coordinate polynomials achieve the maximum degree d.

			# monomials in model $Y = X^{-1}$				# monomials in model $XY = 1$			
k_2	k_1	m	bound	max	min	avg	bound	max	min	avg
1	0	2	2^2	3	2	2.50	5	3	3	3.00
2	1	2	2^3	6	4	5.00	5	3	3	3.00
3	2	2	2^5	18	16	17.00	5	3	3	3.00
4	3	2	2^9	258	256	257.00	5	3	3	3.00
5	4	2	2^{17}	-	-	-	5	3	3	3.00
6	5	2	2^{33}	-	-	-	5	3	3	3.00
2	0	4	2^4+1	10	5	7.25	17	9	5	6.75
3	1	4	2^8+1	70	59	63.75	17	10	5	7.00
4	2	4	$2^{16}+1$	4115	4012	4070.25	17	10	5	7.00
5	3	4	$2^{32}+1$	-	-	-	17	10	5	7.00
6	4	4	$2^{64}+1$	-	-	-	17	10	5	7.00
3	0	8	2^8+1	134	97	114.62	65	30	9	18.00
4	1	8	$2^{16}+1$	16425	15975	16211.88	65	35	9	19.00
5	2	8	$2^{32}+1$	-	-	-	65	35	9	19.00
6	3	8	$2^{64}+1$	-	-	-	65	35	9	19.00

Table 13: Coefficients $C_{j,i,k}$ of the linearized polynomial representation $\mathsf{L}'_j : \mathbb{T}_6^t \to \mathbb{T}_6$ for $j = 0$ of the linear layer $\mathsf{L} : \mathbb{T}_3^{8\times t} \to \mathbb{T}_3^{8\times t}$, cf. Equation (17). Here $0 \le i < t$, $0 \le k < m$.

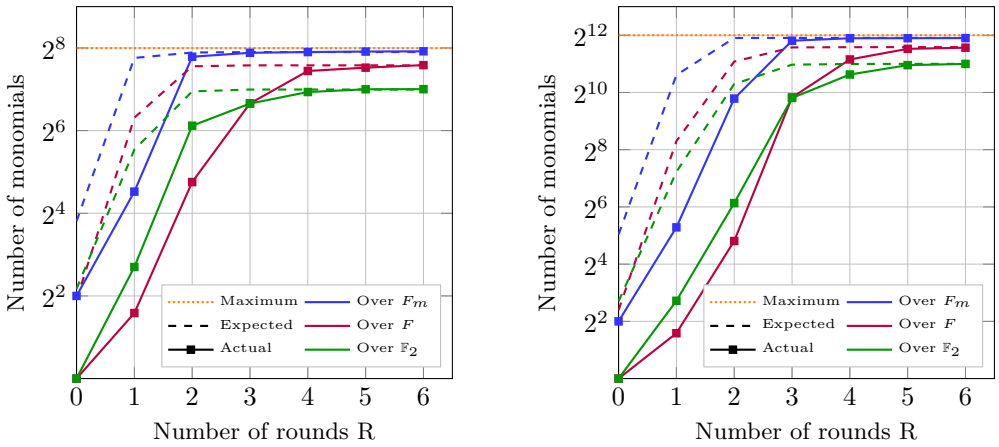
(a) $t = 8$.

$i \setminus k$	0	1	2	3	4	5	6	7
0	0000001400000000	FD48001400000000	1400001400000000	E948001400000000	0000001400000000	FD48001400000000	1400001400000000	E948001400000000
1	1401011001041401	E94826FD49D56E8C	00000104270F0441	FD4827E949D5BBC4	1401011000000000	E94826FD49C187C4	00000104260B0441	FD4827E949C1468C
2	1400010000141400	E94826FD49D56E8C	0000001448FD1401	FD4827E949D5BBC4	1400010000000000	E94826FD49C187C4	0000001448E91401	FD4827E949C1468C
3	0000010401040000	000001140328E948	00000104270F0441	00000114023CFD48	0000010400000000	00000114023CE948	00000104260B0441	000001140328FD48
4	0000001400140000	000001140328E948	0000001448FD1401	00000114023CFD48	0000001400000000	00000114023CE948	0000001448E91401	000001140328FD48
5	1401011000000000	140149E900000000	1401010400000000	140149FD00000000	1401011000000000	140149E900000000	1401010400000000	140149FD00000000
6	1400010000000000	140149E900000000	1400001400000000	140149FD00000000	1400010000000000	140149E900000000	1400001400000000	140149FD00000000
7	0000010400000000	FD48001400000000	1401010400000000	E948001400000000	0000010400000000	FD48001400000000	1401010400000000	E948001400000000

(b) $t = 12$.

$i \setminus k$	0	1	2	3	4	5	6	7
0	0000005D00000000	4ED6005D00000000	5D00005D00000000	13D6005D00000000	0000005D00000000	4ED6005D00000000	5D00005D00000000	13D6005D00000000
1	0000040900000000	B39E004900000000	4904040900000000	FA9E004900000000	0000040900000000	B39E004900000000	4904040900000000	FA9E004900000000
2	4900040000000000	49049AFA00000000	4900004900000000	49049AB300000000	4900040000000000	49049AFA00000000	4900004900000000	49049AB300000000
3	4904044000000000	49049AFA00000000	4904040900000000	49049AB300000000	4904044000000000	49049AFA00000000	4904040900000000	49049AB300000000
4	0000004900490000	0000044908C7FA9E	000000499EB34904	0000044908C7B39E	0000004900000000	0000044908C7FA9E	000000499EFA4904	000004490C8EB39E
5	14010519050D1401	E94822B4455B9412	0000050DA2040DD5	FD4823A04112085A	1401051900000000	E94822B441067D5A	0000050DA7090DD5	FD4823A0454FF512
6	5D00050005D5D00	13D6A74ED3E8716B	0000005DD64E5D05	4ED6A213D3E899BD	5D00050000000000	13D6A74ED3B562BD	0000005DD6135D05	4ED6A213D3B5D76B
7	49040544050D4904	FA9E80A79915F6AF	0000050DA2040DD5	B39E84EE9801DF31	4904054400000000	FA9E80A798480C31	0000050DA7090DD5	B39E84EE995C6CAF
8	0000001400140000	000001140328E948	0000001448FD1401	00000114023CFD48	0000001400000000	00000114023CE948	0000001448E91401	000001140328FD48
9	1401011000000000	140149E900000000	1401010400000000	140149FD00000000	1401011000000000	140149E900000000	1401010400000000	140149FD00000000
10	1400010000000000	140149E900000000	1400001400000000	140149FD00000000	1400010000000000	140149E900000000	1400001400000000	140149FD00000000
11	0000010400000000	FD48001400000000	1401010400000000	E948001400000000	0000010400000000	FD48001400000000	1401010400000000	E948001400000000

A.3 Algebraic Attacks - Interpolation Attack



(a) For $(k_1, k_2, t) = (1, 2, 2)$. **(b)** For $(k_1, k_2, t) = (1, 2, 3)$.

Figure 10: Number of monomials reached in toy RainHash2.0 with $(k_1, k_2) = (1, 2)$.

A.4 Algebraic Attacks – Gröbner Basis Attacks

A.4.1 Estimated Complexity

Table 14: Estimated D_{reg} and $\log_2$ Gröbner basis complexity (in parentheses, $\omega = 2$) under different attack scenarios for RainHash2.0 with $(k_1, k_2) = (3, 6)$ and $m = 8$.

	Sponge				**Compression**	
t	(2nd) preimage	CICO-1	CICO-2	CICO-3	preimage	2nd preimage
8	65 (248)	256 (640)	256 (583)	256 (519)	256 (583)	65 (248)
12	256 (693)	256 (831)	256 (788)	256 (742)	256 (788)	97 (376)

(a) Model over $F = \mathbb{F}_{2^8}$ with degree-q_1 field equations for R = 1. All systems are overdetermined. D_{reg} is computed as the minimum of the Hilbert series estimate (under the semi-regularity assumption) and q_1.

t/R	1	2	3	4	5	6
8	9 (27)	17 (58)	25 (90)	33 (121)	41 (**153**)	49 (185)
12	13 (43)	25 (90)	37 (**137**)	49 (185)	61 (232)	73 (280)

(b) Model over $F_m = \mathbb{F}_{2^{64}}$ at structural degree 2 for $1 \leq \text{R} \leq 6$. All attack scenarios yield identical values of D_{reg} (computed from the Macaulay bound under the regularity assumption), as systems are either square or made square by fixing excess variables.

A.4.2 Experimental Results

All practical experiments are conducted on a machine with an Intel Xeon E5-2630 v3 @ 2.40 GHz and 378 GB RAM under Debian 11 using Magma V2.26-6. Given the large number of configurations, we imposed time and memory limits on every run. Because the cost varies substantially across instances, these limits were adjusted per experiment; the values used are stated in the caption of the respective table. For a run that completes

within its limits, we report the measured Gröbner-basis solving time and peak memory. For a run that does not complete, the entry instead reports the resources it consumed up to termination (the elapsed running time and the peak memory reached) annotated with a marker for the cause: † if it hit the time limit, and ‡ if it ran out of memory (either Magma aborting on a failed allocation, or the job being killed by the system once it exceeded its memory budget). Note that for $(k_1, k_2, m) = (2, 3, 2)$, some runs were executed concurrently on the shared machine (so a memory-pressure termination reflects combined demand), while for $(k_1, k_2, m) = (2, 4, 4)$, jobs ran sequentially against an explicit per-job memory limit of 320 GB. When a run is terminated so early that the corresponding quantity was never recorded (for instance while still computing the degree of regularity, before any memory statistics are emitted) only the marker is shown; the elapsed time, which we reconstruct from the job's start and cancellation timestamps, is still reported where available. A dash (-) marks a configuration we did not run.

We analyze the results along three axes: the attack scenario, the algebraic model, and the growth in the parameters t and m. Throughout, we base quantitative statements on runs that complete within their limits, since the time and memory of an aborted run reflect the phase at which it was terminated (and, for $(2, 3, 2)$, the concurrent load on the machine) rather than the true solving cost; aborted runs are read only as lower bounds confirming that the cost has already left the feasible regime.

- *Attack scenario.* Across all instances and both models, the three scenarios are consistently ordered by difficulty. The *compression second-preimage* system is the easiest to solve: it admits completed runs at the highest round numbers (e.g. for $(2, 3, 2)$ over $\mathbb{F}_2$ up to $\mathrm{R} = 8$ rounds at $t = 2$ and $\mathrm{R} = 5$ rounds at $t = 3$). The *sponge (second-)preimage* system is of intermediate difficulty, completing to round numbers similar to the compression second-preimage case for the *square* state ($t = m$) and falling behind for the *wide* state ($t = 3m/2$). The *compression preimage* system is by far the hardest: it leaves the feasible regime first, already exceeding the time limit at $\mathrm{R} = 3$ for $(2, 3, 2)$ at $t = 2$ in the $\mathbb{F}_2$ model, where the other two scenarios are still solved in well under a second.

- *Algebraic model.* The F and $\mathbb{F}_2$ models trade variable count against field-equation degree (Table 6): relative to the F model, the $\mathbb{F}_2$ model has a factor of $n_2/m = 2^{k_1}$ more variables, but its field equations have degree only 2, versus $q_1 = 2^{2^{k_1}}$ over F. Since we fix $k_1 = 2$ in both test instances, this ratio stays constant at $2^{k_1} = 4$ throughout, so varying k_2 (and hence m) isolates the effect of the extension degree. In general, the $\mathbb{F}_2$ model consistently reaches higher round numbers before becoming infeasible (e.g. for $(2, 3, 2)$ at the square state $t = 2$, the sponge system completes through $\mathrm{R} = 8$ over $\mathbb{F}_2$ but only through $\mathrm{R} = 4$ over F). However, for the larger instance $(2, 4, 4)$ with $t = 6$ both models become infeasible within our resource limits after only one or two rounds, primarily due to memory exhaustion, despite the larger 24 h / 320 GB budget.

- *Growth in t and m.* The feasibility boundary recedes sharply as the state size t and the extension degree m increase. Increasing t at fixed (k_1, k_2, m) enlarges the system and lowers the highest solvable round: for $(2, 3, 2)$, the sponge scenario completes through $\mathrm{R} = 8$ at $t = 2$ but only through $\mathrm{R} \leq 2$ at $t = 3$. Increasing m has an even stronger effect: for the larger instance $(2, 4, 4)$, essentially no configuration completes beyond $\mathrm{R} = 2$, and most scenarios already exceed the (now much larger, 24 h and 320 GB) budget at one or two rounds.

Table 15: Time and memory of Gröbner basis computation for RainHash2.0 with $(k_1, k_2, m) = (2, 3, 2)$, i.e., $F = \mathbb{F}_{2^4}$ and $F_m = \mathbb{F}_{2^8}$. Here $\chi = 1$.

		Sponge (2nd) preimage		Compression preimage		Compression 2nd preimage	
Model	R	Time (s)	Mem (GB)	Time (s)	Mem (GB)	Time (s)	Mem (GB)
$\mathbb{F}_2$	1	0.5	0.03	0.5	0.03	0.5	0.03
	2	0.5	0.03	38.0	0.66	0.5	0.03
	3	0.5	0.03	43 227.0†	131.50	0.5	0.03
	4	0.5	0.03	5246.4	65.06‡	0.5	0.03
	5	47.1	0.24	9328.2	28.80‡	46.7	0.24
	6	124.3	0.67	17 072.6	17.33‡	129.2	0.67
	7	173.0	0.90	40 076.4	98.01‡	166.8	0.90
	8	364.7	1.61	20 730.0	27.40‡	362.2	1.61
	9	7504.0	20.65‡	9561.8	26.90‡	2516.1	15.98‡
F	1	0.5	0.03	0.5	0.03	0.5	0.03
	2	0.5	0.03	22.9	0.39	0.5	0.03
	3	0.5	0.06	6656.2	16.75	0.5	0.06
	4	1.4	0.14	43 205.0†		1.4	0.14
	5	230.0	14.72‡	4471.3	19.83‡	232.1	14.72‡
	6	1909.0	41.36‡	11 321.5	113.45‡	1174.3	31.58‡
	7	7969.4	34.65‡	3363.6	28.49‡	19 643.0	98.18‡
	8	1954.2	24.94‡	5343.9	30.82‡	2716.5	26.95‡
	9	43 226.0†		9004.2	25.15‡	1110.7	10.47‡

† Time limit of $12\,h = 43200\,s$ reached; report running time and reached peak memory.
‡ Out of memory (no explicit limit set); report running time and reached peak memory.

(a) State size $t = 2$.

		Sponge (2nd) preimage		Compression preimage		Compression 2nd preimage	
Model	R	Time (s)	Mem (GB)	Time (s)	Mem (GB)	Time (s)	Mem (GB)
$\mathbb{F}_2$	1	0.6	0.03	43 203.0†		0.5	0.03
	2	34 166.9	29.64	5241.1	80.55‡	0.5	0.03
	3	86 417.0†		17 107.0	147.53‡	1.1	0.03
	4	8491.0	14.61‡	8489.9	22.36‡	211.8	0.77
	5	2672.8	18.13‡	3489.5	24.53‡	158.7	1.62
	6	43 220.0†		43 220.0†		2586.5	15.55‡
F	1	0.5	0.03	43 227.0†	51.91	0.5	0.03
	2	620.3	4.24	17 507.1	138.67‡	0.5	0.03
	3	27 174.6	157.98‡	1629.3	22.54‡	4.4	0.34
	4	2653.9	34.02‡	10 863.1	92.68‡	1264.5	40.23‡
	5	3332.8	65.42‡	267.9	6.59‡	43 222.0†	

† Time limit of $12\,h = 43200\,s$ reached; report running time and reached peak memory.
‡ Out of memory (no explicit limit set); report running time and reached peak memory.

(b) State size $t = 3$.

Table 16: Time and memory of Gröbner basis computation for RainHash2.0 with $(k_1, k_2, m) = (2, 4, 4)$, i.e., $F = \mathbb{F}_{2^4}$ and $F_m = \mathbb{F}_{2^{16}}$. Here $\chi = 2$.

		Sponge (2nd) preimage		Compression preimage		Compression 2nd preimage	
Model	R	Time (s)	Mem (GB)	Time (s)	Mem (GB)	Time (s)	Mem (GB)
$\mathbb{F}_2$	1	0.5	0.03	23 585.7	‡	0.5	0.03
	2	36.7	0.06	3735.0	60.80‡	31.4	0.06
	3	-	-	-	-	-	-
F	1	0.5	0.03	86 405.0†		0.5	0.03
	2	†		2172.7	29.39‡	14 428.4	‡
	3	1790.3	40.15‡	-	-	664.5	11.66‡

† Time limit of $24\,h = 86400\,s$ reached; report running time and reached peak memory.
‡ Out of memory (320 GB limit); report running time and reached peak memory.

(a) State size $t = 4$.

		Sponge (2nd) preimage		Compression preimage		Compression 2nd preimage	
Model	R	Time (s)	Mem (GB)	Time (s)	Mem (GB)	Time (s)	Mem (GB)
$\mathbb{F}_2$	1	9114.3	‡	7649.5	‡	0.5	0.03
	2	4791.8	82.84‡		‡	86 429.0†	
F	1	63 301.3	336.47‡	7064.9	‡	1.4	0.10
	2	43 216.0†		1432.3	25.20‡	1961.0	‡

† Time limit of $24\,h = 86400\,s$ reached; report running time and reached peak memory.
‡ Out of memory (320 GB limit); report running time and reached peak memory.

(b) State size $t = 6$.

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020. doi:10.13154/tosc.v2020.i3.1-45.

[ABK+23] Tomer Ashur, Amit Singh Bhati, Al Kindi, Mohammad Mahzoun, and Léo Perrin. XHash: Efficient STARK-friendly hash function. Cryptology ePrint Archive, Report 2023/1045, 2023. URL: https://eprint.iacr.org/2023/1045.

[ABK+26] Tomer Ashur, Amit Singh Bhati, Al Kindi, Mohammad Mahzoun, Léo Perrin, and Sundas Tariq. The XHash family for ZK-friendly hash functions. *DCC*, 94(132), 2026. doi:10.1007/s10623-026-01875-1.

[AD18] Tomer Ashur and Siemen Dhooghe. MARVELlous: a STARK-friendly family of cryptographic primitives. Cryptology ePrint Archive, Report 2018/1098, 2018. URL: https://eprint.iacr.org/2018/1098.

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 151–171. Springer, Cham, September 2019. doi:10.1007/978-3-030-29962-0_8.

[AMD25] AMD, Inc. Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide (UG953), 2025. Accessed on 07 June 2025. URL: https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/LUT4.

[AMPŠ24] Tomer Ashur, Mohammad Mahzoun, Jim Posen, and Danilo Šijačić. Vision Mark-32: ZK-friendly hash function over binary tower fields. Cryptology ePrint Archive, Report 2024/633, 2024. URL: https://eprint.iacr.org/2024/633.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson, Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 2013*, volume 7954 of *LNCS*, pages 119–135. Springer, Berlin, Heidelberg, June 2013. doi:10.1007/978-3-642-38980-1_8.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_17.

[Bak25] Antoine Bak. A practical distinguisher on the full Skyscraper permutation. Cryptology ePrint Archive, Report 2025/102, 2025. URL: https://eprint.iacr.org/2025/102.

[BBB+25a] Antoine Bak, Augustin Bariant, Aurélien Boeuf, Pierre Briaud, Morten Øygarden, and Atharva Phanse. The algebraic CheapLunch: Extending FreeLunch attacks on arithmetization-oriented primitives beyond CICO-1. Cryptology ePrint Archive, Report 2025/2040, 2025. URL: https://eprint.iacr.org/2025/2040.

[BBB+25b] Augustin Bariant, Aurélien Boeuf, Pierre Briaud, Maël Hostettler, Morten Øygarden, and Håvard Raddum. Improved resultant attack against arithmetization-oriented primitives. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 335–367. Springer, Cham, August 2025. doi:10.1007/978-3-032-01901-1_11.

[BBB+25c] Carsten Baum, Ward Beullens, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, et al. FAEST v2: Algorithm specifications. Technical report, National Institute of Standards and Technology, 2025. Round 2 Submission to the NIST PQC Additional Digital Signatures Call. https://faest.info/faest-spec-v2.0.pdf.

[BBB+25d] Carsten Baum, Ward Beullens, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. Shorter, tighter, FAESTer: Optimizations and improved (QROM) analysis for VOLE-in-the-head signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 124–156. Springer, Cham, August 2025. doi:10.1007/978-3-032-01887-8_5.

[BBB+26] Carsten Baum, Marvin Beckmann, Ward Beullens, Shibam Mukherjee, and Christian Rechberger. Concretely efficient blind signatures based on VOLE-in-the-head proofs and the MAYO trapdoor. Cryptology ePrint Archive, Paper 2026/109, 2026. URL: https://eprint.iacr.org/2026/109.

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and Jive compression mode. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 507–539. Springer, Cham, August 2023. doi:10.1007/978-3-031-38548-3_17.

[BBD+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_19.

[BBdSG+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Sebastian Ramacher, Christian Rechberger, Emmanuela Orsini, Lawrence Roy, et al. FAEST: Algorithm specifications. Technical report, National Institute of Standards and Technology, 2023. Round 1 Submission to the NIST PQC Additional Digital Signatures Call. https://faest.info/faest-spec-v1.1.pdf.

[BBHN26] Antoine Bak, Augustin Bariant, M Hostettler, and Vincent Neiger. Resultants meet resultant: Improving CICO-1 and CICO-2 attacks on ZK-friendly permutations. Cryptology ePrint Archive, Paper 2026/1281, 2026. URL: https://eprint.iacr.org/2026/1281.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP*

2018, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

[BBL+24] Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IV*, volume 14923 of *LNCS*, pages 139–173. Springer, Cham, August 2024. doi:10.1007/978-3-031-68385-5_5.

[BBLP22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. Algebraic attacks against some arithmetization-oriented primitives. *IACR Trans. Symm. Cryptol.*, 2022(3):73–101, 2022. doi:10.46586/tosc.v2022.i3.73-101.

[BBM+24] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 463–493. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0875-1_15.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 12–23. Springer, Berlin, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_2.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and Luffa. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 252–269. Springer, Berlin, Heidelberg, February 2011. doi:10.1007/978-3-642-21702-9_15.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 299–328. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_11.

[BDH+23] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. TurboSHAKE. Cryptology ePrint Archive, Report 2023/342, 2023. URL: https://eprint.iacr.org/2023/342.

[BDPA11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. URL: https://keccak.team/files/CSF-0.1.pdf.

[BDPA11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference. SHA-3 competition (round 3), 2011. URL: https://keccak.team/files/Keccak-reference-3.0.pdf.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the Sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Berlin, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_11.

[BDPV13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_19.

[BDSW23] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector OLE-based zero-knowledge protocols. *DCC*, 91(11):3527–3561, 2023. doi:10.1007/s10623-023-01292-8.

[BFS04] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *ICPSS 2004*, pages 71–75, Paris, France, 2004. URL: http://magali.bardet.free.fr/Publis/ltx43BF.pdf.

[BGK+25a] Clémence Bouvier, Lorenzo Grassi, Dmitry Khovratovich, Katharina Koschatko, Christian Rechberger, Fabian Schmid, and Markus Schofnegger. Skyscraper: Fast hashing on big primes. *IACR TCHES*, 2025(2):743–780, 2025. doi:10.46586/tches.v2025.i2.743-780.

[BGK+25b] Clémence Bouvier, Lorenzo Grassi, Dmitry Khovratovich, Katharina Koschatko, Christian Rechberger, Fabian Schmid, and Markus Schofnegger. Skyscraper-v2: Fast hashing on big primes. Cryptology ePrint Archive, Report 2025/058, 2025. URL: https://eprint.iacr.org/2025/058.

[BJGP25] Antoine Bak, Guilhem Jazeron, Pierre Galissant, and Léo Perrin. Attacking split-and-lookup-based primitives using probabilistic polynomial system solving: Applications to round-reduced Monolith and full-round Skyscraper. *IACR Trans. Symm. Cryptol.*, 2025(3):337–367, 2025. doi:10.46586/tosc.v2025.i3.337-367.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Berlin, Heidelberg, September 2007. doi:10.1007/978-3-540-74735-2_31.

[BOS+26] Carsten Baum, Emmanuela Orsini, Peter Scholl, Benoit Razet, Marcella Hastings, Shibam Mukherjee, Christian Rechberger, and James Parker. Schmivitz: VOLEitH based ZK gadgets for threshold cryptography. Preview writeup, National Institute of Standards and Technology, 2026. Submission to the NIST Threshold Call (NIST IR 8214C). https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/Schmivitz-PW01.pdf.

[BP25] Antoine Bak and Léo Perrin. On the security of split-and-lookup-based ZK-friendly primitives. *IACR Trans. Symm. Cryptol.*, 2025(2):87–123, 2025. doi:10.46586/tosc.v2025.i2.87-123.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *DCC*, 70(3):369–383, 2014. doi:10.1007/s10623-012-9697-z.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Berlin, Heidelberg, August 1991. doi:10.1007/3-540-38424-3_1.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 245–259. Springer, Berlin, Heidelberg, March 1999. doi:10.1007/3-540-48519-8_18.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017. doi:10.1145/3133956.3133997.

[CG23] Alessio Caminata and Elisa Gorla. Solving degree, last fall degree, and related invariants. *J. Symb. Comput.*, 114:322–335, 2023. doi:10.1016/j.jsc.2022.05.001.

[CGG+22] Carlos Cid, Lorenzo Grassi, Aldo Gunsing, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Influence of the linear layer on the algebraic degree in SP-networks. *IACR Trans. Symm. Cryptol.*, 2022(1):110–137, 2022. doi:10.46586/tosc.v2022.i1.110-137.

[CR25] Luca Campa and Arnab Roy. Gröbner basis cryptanalysis of Anemoi. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part I*, volume 15601 of *LNCS*, pages 303–332. Springer, Cham, May 2025. doi:10.1007/978-3-031-91107-1_11.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021. doi:10.1007/s00145-021-09398-9.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165. Springer, Berlin, Heidelberg, January 1997. doi:10.1007/BFb0052343.

[DKR+22] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 843–857. ACM Press, November 2022. doi:10.1145/3548606.3559353.

[DP24] Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, Report 2024/504, 2024. URL: https://eprint.iacr.org/2024/504.

[DP25] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 93–122. Springer, Cham, May 2025. doi:10.1007/978-3-031-91134-7_4.

[DR02a] Joan Daemen and Vincent Rijmen. AES and the wide trail design strategy (invited talk). In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 108–109. Springer, Berlin, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_7.

[DR02b] Joan Daemen and Vincent Rijmen. Security of a wide trail design (invited talk). In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT 2002*, volume 2551 of *LNCS*, pages 1–11. Springer, Berlin, Heidelberg, December 2002. doi:10.1007/3-540-36231-2_1.

[Dwo15] Morris J. Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication FIPS PUB 202, National Institute of Standards and Technology, 2015. https://doi.org/10.6028/NIST.FIPS.202.

[EGL+20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. An algebraic attack on ciphers with low-degree round functions: Application to full MiMC. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 477–506. Springer, Cham, December 2020. doi:10.1007/978-3-030-64837-4_16.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999. doi:10.1016/S0022-4049(99)00005-5.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *ISSAC 2002*, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/780506.780516.

[FGLM93] Jean-Charles Faugère, Patrizia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993. doi:10.1006/jsco.1993.1051.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005. doi:10.1007/11535218_10.

[FM17] Jean-Charles Faugère and Chenqi Mou. Sparse FGLM algorithms. *J. Symb. Comput.*, 80:538–569, 2017. doi:10.1016/j.jsc.2016.07.025.

[FP97] John L. Fan and Christof Paar. On efficient inversion in tower fields of characteristic two. In *Proceedings of IEEE International Symposium on Information Theory*, pages 20–, 1997. doi:10.1109/ISIT.1997.612935.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[Gam23] Onur Berkay Gamgam. An unrolled and pipelined architecture for SHA2 family of hash functions on FPGA. In *2023 16th International Conference on Information Security and Cryptology (ISCTürkiye)*, pages 1–6, 2023. doi:10.1109/ISCTrkiye61151.2023.10336096.

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. Horst meets fluid-SPN: Griffin for zero-knowledge applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 573–606. Springer, Cham, August 2023. doi:10.1007/978-3-031-38548-3_19.

[GKK+26] Lorenzo Grassi, Dmitry Khovratovich, Katharina Koschatko, Christian Rechberger, Markus Schofnegger, Verena Schröppel, and Zhuo Wu. Poseidon(2)b: Binary field versions of Poseidon/Poseidon2. *CiC*, 2(4):15, 2026. doi:10.62056/a66ce0zn4.

[GKL+24] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations. *IACR Trans. Symm. Cryptol.*, 2024(3):44–83, 2024. doi:10.46586/tosc.v2024.i3.44-83.

[GKM+09] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. In *Symmetric Cryptography, 11.01. – 16.01.2009*, volume 09031 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. URL: http://drops.dagstuhl.de/opus/volltexte/2009/1955/.

[GKR+19] Lorenzo Grassi, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458, 2019. URL: https://eprint.iacr.org/archive/2019/458/20200205:104144.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the Poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203. Springer, Cham, July 2023. doi:10.1007/978-3-031-37679-5_8.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 674–704. Springer, Cham, May 2020. doi:10.1007/978-3-030-45724-2_23.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 18–37. Springer, Berlin, Heidelberg, March 2015. doi:10.1007/978-3-662-46706-0_2.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985. doi:10.1145/22145.22178.

[GOPS22] Lorenzo Grassi, Silvia Onofri, Marco Pedicini, and Luca Sozzi. Invertible quadratic non-linear layers for MPC-/FHE-/ZK-friendly schemes over $\mathbb{F}_p^n$: Application to Poseidon. *IACR Trans. Symm. Cryptol.*, 2022(3):20–72, 2022. doi:10.46586/tosc.v2022.i3.20-72.

[Gra18] Lorenzo Grassi. Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. *IACR Trans. Symm. Cryptol.*, 2018(2):133–160, 2018. doi:10.13154/tosc.v2018.i2.133-160.

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. A new structural-differential property of 5-round AES. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 289–317. Springer, Cham, April / May 2017. doi:10.1007/978-3-319-56614-6_10.

[GRS21] Lorenzo Grassi, Christian Rechberger, and Markus Schofnegger. Proving resistance against infinitely long subspace trails: How too choose the linear layer. *IACR Trans. Symm. Cryptol.*, 2021(2):314–352, 2021. doi:10.46586/tosc.v2021.i2.314-352.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

[HKPR25] Florian Hirner, Florian Krieger, Constantin Piber, and Sujoy Sinha Roy. Accelerating hash-based polynomial commitment schemes with linear prover time. *IACR TCHES*, 2025(4):341–385, 2025. doi:10.46586/tches.v2025.i4.341-385.

[HKY15] Ming-Deh A. Huang, Michiel Kosters, and Sze Ling Yeo. Last fall degree, HFE, and Weil descent attacks on ECDLP. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 581–600. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-47989-6_28.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 28–40. Springer, Berlin, Heidelberg, January 1997. doi:10.1007/BFb0052332.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154. Springer, Berlin, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_13.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 - Efficient short-input hashing for post-quantum applications. *IACR Trans. Symm. Cryptol.*, 2016(2):1–29, 2016. URL: https://tosc.iacr.org/index.php/ToSC/article/view/563, doi:10.13154/tosc.v2016.i2.1-29.

[KN10] Dmitry Khovratovich and Ivica Nikolic. Rotational cryptanalysis of ARX. In Seokhie Hong and Tetsu Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 333–346. Springer, Berlin, Heidelberg, February 2010. doi:10.1007/978-3-642-13858-4_19.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 196–211. Springer, Berlin, Heidelberg, December 1995. doi:10.1007/3-540-60590-8_16.

[KR21] Nathan Keller and Asaf Rosemarin. Mind the middle layer: The HADES design strategy revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 35–63. Springer, Cham, October 2021. doi:10.1007/978-3-030-77886-6_2.

[KSK25] Ruby Kumari, Sumeet Saurav, and Abhijit Karmakar. High-performance FPGA implementation of a recursive modular Karatsuba multiplier over $GF(2^m)$. In Ratna Dutta, Luca De Feo, and Sugata Gangopadhyay, editors, *INDOCRYPT 2025*, volume 16372 of *LNCS*, pages 21–41. Springer, Cham, December 2025. doi:10.1007/978-3-032-13301-4_2.

[LBM25] Charlotte Lefevre, Mario Marhuenda Beltrán, and Bart Mennink. To pad or not to pad? Padding-free arithmetization-oriented sponges. *IACR Trans. Symm. Cryptol.*, 2025(1):97–137, 2025. doi:10.46586/tosc.v2025.i1.97-137.

[LMM24] Fukang Liu, Mohammad Mahzoun, and Willi Meier. Modelling ciphers with overdefined systems of quadratic equations: Application to Friday, Vision, RAIN and Biscuit. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VII*, volume 15490 of *LNCS*, pages 424–456. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0941-3_14.

[LMR+09] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound distinguishers: Results on the full Whirlpool compression function. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 126–143. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_8.

[Mat85] Stephen M Matyas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Berlin, Heidelberg, May 1994. doi:10.1007/3-540-48285-7_33.

[McL25] Michael B. McLoughlin. addchain: Cryptographic addition chain generation. GitHub, 2025. https://github.com/mmcloughlin/addchain, Accessed in December 2025.

[MRST09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 260–276. Springer, Berlin, Heidelberg, February 2009. doi:10.1007/978-3-642-03317-9_16.

[Nat15] National Institute of Standards and Technology. Secure Hash Standard (SHS). Federal Information Processing Standards Publication FIPS PUB 180-4, National Institute of Standards and Technology, 2015. https://doi.org/10.6028/NIST.FIPS.180-4.

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Report 2020/1143, 2020. URL: https://eprint.iacr.org/2020/1143.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. URL: https://api.semanticscholar.org/CorpusID:121656251.

[Wie88] Doug Wiedemann. An iterated quadratic extension of GF(2). *The Fibonacci Quarterly*, 4(26):290–295, 1988.

[WXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *ACM-SIAM Symposium on Discrete Algorithms*, 2023. URL: https://api.semanticscholar.org/CorpusID:259937341.

[Xil22] Xilinx. DMA/Bridge Subsystem for PCI Express v4.1 Product Guide (PG195), 2022. Accessed on 23 March 2026. URL: https://www.amd.com/content/dam/xilinx/support/documents/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf.

[Xil23] Xilinx. Alveo u280 data center accelerator card data sheet (ds963), 2023. Accessed on 07 June 2025. URL: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021. doi:10.1145/3460120.3484556.

[YZY+24] Hong-Sen Yang, Qun-Xiong Zheng, Jing Yang, Quan feng Liu, and Deng Tang. A new security evaluation method based on resultant for arithmetic-oriented algorithms. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VII*, volume 15490 of *LNCS*, pages 457–489. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0941-3_15.

[YZY25] Hong-Sen Yang, Qun-Xiong Zheng, and Jing Yang. Algebraic cryptanalysis of AO primitives based on polynomial decomposition: Applications to Rain and full AIM-I/III/V. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part I*, volume 16245 of *LNCS*, pages 383–410. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5018-0_13.