pSquare-hash: A Family of Tweakable Hash Functions for Physically Secure PQ Signatures

Lorenzo Grassi[1], Mario Marhuenda-Beltrán[2], Thorben Moos[3], Fabian Schmid[4], Matthias Johann Steiner[5], and Hailun Yan[5]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Radboud University, Nijmegen, The Netherlands
[3] UCLouvain, Louvain-la-Neuve, Belgium
[4] Graz University of Technology, Graz, Austria
[5] Freischaffender Forscher, Klagenfurt am Wörthersee, Austria
[6] University of Chinese Academy of Sciences, Beijing, China

Abstract In 2020 and 2024 respectively, NIST released a Special Publication (SP 800-208) and a Federal Information Processing Standard (FIPS 205) specifying hash-based signature schemes with natural quantum resistance thanks to their symmetric foundation. The former recommends the stateful hash-based signature schemes LMS and XMSS, whereas the latter standardizes their stateless counterpart SPHINCS$^+$. While in principle all three constructions can be instantiated with any secure cryptographic hash function, the concrete instances recommended by NIST are currently limited to either the SHA-2 or the SHA-3 family. Building on the maturity of these standardized families is of course a sensible choice. Yet, we argue that neither is particularly well suited for this purpose, especially once physical security matters. As an alternative we suggest pSquare-hash, an arithmetization-oriented family of lightweight *tweakable* hash functions. We demonstrate that such dedicated tweakable constructions ideally suit the instantiation and security requirements of hash-based signature schemes, potentially leading to efficiency advantages over standard concatenation-based approaches through either a reduction of the permutation size or the number of calls. With respect to physical security, the presence of the tweak enables a clean separation between inputs that need to be protected against leakage/faults and those that are insensitive. The arithmetization-oriented nature and choice of prime enable the effective utilization of masking schemes with superior passive and active attack resistance (e.g., prime-field and/or inner-product masking) and keep the design suitable for zero-knowledge applications. We compare higher-order masked software (Cortex-M4) and hardware (NanGate 15 nm) implementations of pSquare-hash to equivalent SHA-2, SHA-3, `SKINNY-Hash`, `Ascon-Hash` and Poseidon2 instances and exhibit favorable characteristics whenever masking is applied.

1 Introduction

Digital signatures provide mathematical assurance that messages transmitted over a network indeed originate from the claimed sender and have not been

modified since. Such cryptographic assurances are critical to enable a dependable exchange of information over insecure communication channels. Digital signature schemes are known since the advent of public-key cryptography in the 1970s. Traditionally, such schemes largely relied on either the hardness of *integer factorization* (e.g., RSA) or *discrete logarithms* in finitely generated groups (e.g., Diffie-Hellman, ElGamal, Schnorr). However, in his seminal work [108], Shor proposed quantum algorithms that solve both problems in polynomial time. Hence, if quantum computers of sufficient size and quality to run Shor's algorithm efficiently can be realized in the future, traditional public-key infrastructures will become insecure. To address this problem, the National Institute of Standards and Technology (NIST) launched a Post-Quantum Cryptography (PQC) standardization competition in 2016.[7] In 2022, the first group of selected winners was announced: CRYSTALS-Kyber [29, 107] (key encapsulation), CRYSTALS-Dilithium [48, 84] (signature), FALCON [104] (signature) and SPHINCS$^+$ [17, 18, 69] (signature). In 2024, selected instances of Kyber, Dilithium and SPHINCS$^+$ were standardized as ML-KEM (FIPS 203 [100]), ML-DSA (FIPS 204 [99]) and SLH-DSA (FIPS 205 [101]), respectively, while the FALCON standard is still under development at this time. Three out of the four selected algorithms rely on the hardness of lattice problems, leaving SPHINCS$^+$ as the only algorithm whose security is solely based on hash function properties.

1.1 State of the Art

Hash-Based Signatures. Hash-based signature schemes have a long tradition in cryptography with the first scheme being introduced by Lamport in 1979 [81]. Conceptually, all hash-based signature schemes reduce to the same idea: The public key pk is derived from the secret key sk via a deterministic sequence of hashes. To sign a message m, parts of the hash chain between sk and pk are released as signature σ. For verification one checks that σ can be hashed to pk. A many-times signature scheme can be derived from a one-time signature scheme (e.g., a variant of the Winternitz One-Time Signature Scheme (WOTS) [46, 90]) by generating multiple instances of the latter and compressing their public keys via Merkle trees.

Before the first results of the main PQC standardization competition had been determined, NIST put forward a special publication (SP 800-208 [98]) recommending two *stateful* hash-based many-times signature schemes: the Leighton-Micali Signature (LMS) system [88] and the eXtended Merkle Signature Scheme (XMSS) [33, 67]. Like all hash-based signatures, these constructions are inherently resistant to quantum attacks thanks to their reliance on standard model properties of the employed hash function families only. Yet, unlike conventional signature schemes, both LMS and XMSS are stateful, meaning that signing requires to keep track of the current state of the system as part of the secret key to make sure that no internal one-time signing key is ever reused. This requirement obviously narrows the scope in which such schemes can be used. Technically, they

[7] https://csrc.nist.gov/projects/post-quantum-cryptography

do not even fit the standard definition of signatures in cryptography. However, stateful hash-based signature schemes can also be used to construct stateless ones. SPHINCS$^+$ for example is based on variants of the many-times signature scheme XMSS, the few-times signature scheme FORS [18] and the one-time signature scheme WOTS (within XMSS). Internally, the construction requires a *tweakable hash function*

$$\mathsf{TX} : \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}, \tag{1}$$

which takes a public parameter $p \in \mathcal{P}$, a tweak $t \in \mathcal{T}$ and a message $m \in \mathcal{M}$ as input. The public parameter and tweak are used for domain separation and to mitigate multi-target attacks, essentially "uniquifying" every hash function call in the signature scheme [18, 68, 70]. Evidently, the concrete instantiations of SPHINCS$^+$ and SLH-DSA were unable to make use of native tweakable hash functions, simply because such primitives did not exist at the time. Instead, regular hash functions from the standardized SHA-2 [96] and SHA-3 [97] families are typically used[8] by concatenating the tweak with all other inputs before hashing. Such an approach requires either a sufficiently large rate to digest all inputs at once or requires multiple calls to the underlying compression or permutation function. This naturally begs the question whether dedicated tweakable hash functions could improve upon this situation. Exploring this direction obviously necessitates a formal study of the impact of tweakable constructions on the provable security properties of hash-based signature schemes.

Physical Security of PQC Schemes. The physical security of a cryptographic implementation refers (among other aspects) to its resistance to side-channel analysis, fault injection attacks and combinations thereof. While the majority of research works on such attacks has historically focused on symmetric cryptographic primitives (due to their prevalence on embedded devices), implementation attacks also concern public key cryptosystems. Signature schemes, including post-quantum digital signature algorithms, are especially relevant, as their implementations are supposed to frequently sign publicly visible messages under the same private key. Unfortunately, lattice-based post-quantum signature algorithms, including the ones selected for standardization (CRYSTALS-Dilithium and FALCON), rely to a large extent on operations that are notoriously hard to protect against implementation attacks such as rejection sampling, transitions between Boolean and arithmetic representations or floating point arithmetic [5, 38]. Hash-based signature schemes have a notable advantage in this regard. Their main underlying component is a single hash function, and in principle any cryptographically secure hash function can be used to instantiate them. On the one hand, this is beneficial because research on the implementation security of hash functions is significantly more advanced than research on the effective protection of lattice-based cryptography. On the other hand, this enables cryptographers to select or design new hash functions which are specifically tailored towards facilitating the efficient application of strong countermeasures against physical attacks.

[8] The SPHINCS$^+$-Haraka [77] variant was not adopted in SLH-DSA.

1.2 Our Contributions

In this work, we propose pSquare-hash, an arithmetization-oriented family of lightweight *tweakable* hash functions specifically designed for hash-based signature schemes. We demonstrate that such dedicated tweakable constructions can lead to efficiency advantages over standard approaches through either a reduction of the permutation size or the number of calls. We also recall the significant physical security benefits associated with masking cryptographic primitives operating over sufficiently large prime-order fields. Our dedicated hash function and its integration into hash-based signature schemes is then a combination of several novel contributions.

Provable Security: Tweakable Sponge and Compression. As a first contribution, we formally introduce the tweakable sponge TS and the tweakable compression TC. TS and TC are respectively a sponge construction and a compression function instantiated with a tweakable permutation, which can be set up via the `TWEAKEY` framework proposed in [73]. In Section 3, we discuss that TS is indifferentiable from a random oracle up to the birthday bound in the capacity, as well as collision, preimage, second preimage and multi-target collision resistant. TC, albeit not indifferentiable, enjoys good preimage and second preimage resistance as well. Multi-target collision resistance can be seen as a weakened form of collision resistance, resulting in equally promising security bounds.

Symmetric Design: Tweakable Permutations. As a second contribution we propose concrete instances of our generic constructions. In particular, we introduce pSquare-hash in Section 4, a family of lightweight tweakable hash functions based on tweakable permutations defined over $\mathbb{F}^8_{2^{31}-1}$, $\mathbb{F}^{12}_{2^{31}-1}$, $\mathbb{F}^{16}_{2^{31}-1}$ and $\mathbb{F}^{24}_{2^{31}-1}$. pSquare-hash is arithmetization-oriented and masking friendly, operating over the 31-bit Mersenne prime field. Its round function solely relies on squaring as a source of non-linearity and facilitates the efficient application of strong countermeasures against physical attacks. The round numbers we propose are supported by an in-depth statistical and algebraic cryptanalysis given in Section 5.

Performance Evaluation: pSquare-hash vs. SHA-2, SHA-3, `SKINNY-Hash`, `Ascon-Hash` and Poseidon2. To demonstrate the promising efficiency of our designs we benchmark unprotected and higher-order masked implementations of pSquare-hash in software (up to 32 shares) and hardware (up to 4 shares). A comparison to equivalent SHA-2, SHA-3, `SKINNY-Hash` [13], `Ascon-Hash` [44] and Poseidon2 [61, 62] instances is given in Section 6. We study both, plain permutation functions as well as their integration for hashing in SPHINCS$^+$ and find competitive instantiation and performance characteristics when masking is applied. Importantly, due to the chosen algebraic setting, protected instances of pSquare-hash are expected to provide physical security advantages exceeding the ones of existing (lightweight) hash functions by a significant margin.

Table 1. Average number of calls to the underlying Keccak-f[1600] permutation made by SLH-DSA-SHAKE's internal functions under random keys and (n-byte) messages.

Instance	Alg.	$\mathbf{F}$	$\mathbf{PRF}$	$\mathbf{PRF}_{msg}$	$\mathbf{H}$	$\mathbf{H}_{msg}$	$\mathbf{T}_l$	Total
SLH-DSA-SHAKE-128f	Gen	4200 *(93%)*	280 *(6%)*	0 *(0%)*	7 *(0%)*	0 *(0%)*	40 *(1%)*	4527
	Sign	94245 *(89%)*	8272 *(8%)*	1 *(0%)*	2230 *(2%)*	1 *(0%)*	880 *(1%)*	105629
	Vrfy	5907 *(94%)*	0 *(0%)*	0 *(0%)*	264 *(4%)*	1 *(0%)*	115 *(2%)*	6287
SLH-DSA-SHAKE-256f	Gen	16080 *(92%)*	1072 *(6%)*	0 *(0%)*	15 *(0%)*	0 *(0%)*	272 *(2%)*	17439
	Sign	290775 *(83%)*	36144 *(10%)*	1 *(0%)*	18136 *(5%)*	1 *(0%)*	4616 *(1%)*	349673
	Vrfy	8632 *(93%)*	0 *(0%)*	0 *(0%)*	383 *(4%)*	1 *(0%)*	298 *(3%)*	9314

2 A Tweakable Hash Function for Physical Security: The Design Space for pSquare-hash

Before presenting our provable security analysis and specifying our tweakable permutations, we aim to clarify and justify the targeted design space. In essence, there are two independent arguments our proposal tries to make for its high-level design choices. One is about implementation *efficiency*, the other is about implementation *security*.

Implementation Efficiency. Similar to other hash-based signature schemes, SLH-DSA defines multiple internal functions for different purposes [101, §4.1], $\mathbf{F}$, $\mathbf{PRF}$, $\mathbf{PRF}_{msg}$, $\mathbf{H}$, $\mathbf{H}_{msg}$, and $\mathbf{T}_l$. An exemplary breakdown of their respective share of the total number of underlying permutation calls made in SLH-DSA-SHAKE is given in Table 1. Since these internal functions are eventually instantiated via the same underlying hash algorithm in concrete instances (e.g., Keccak/SHA-3 for SLH-DSA-SHAKE), they mainly determine which inputs are digested in what order. Most of them digest public inputs (e.g., seeds, addresses or the initial message) and private ones (e.g., secret keys or chaining elements) together without making much of a distinction. Once physical security matters, this is generally ill-advised. Then, private inputs typically need to be protected against leakage and faults with countermeasures such as masking and redundancy, while public ones do not necessarily require the same level of protection. Hence, some form of input separation is usually advisable in order to optimize different parts of the implementation to the specific needs of the processed data. Alternatively, one may treat all intermediates, both public and private ones, as sensitive and protect all functions uniformly against passive and active adversaries. This is currently the standard approach for SLH-DSA implementations protected against side-channel [106] and fault attacks [58]. However, such a strategy obviously comes at significant, potentially unnecessary, cost and performance overheads. Our implementation *efficiency* argument is therefore that whenever physical security is required, separating public and private inputs in order to process the former in unprotected (e.g., an unmasked tweak schedule) and the latter in protected (e.g. a masked round function) data paths or function calls, can lead to strong efficiency gains over any strategy making no such distinction. When masking is employed as a side-channel and/or partial fault countermea-

sure (usually paired with redundancy to effectively thwart active attacks), such efficiency gains typically increase quadratically in the protection order.

The lack of input separation in existing instances is by no means a necessity. SLH-DSA actually does separate public and private inputs internally. As mentioned before, it is built upon tweakable hash functions as underlying primitives, where the public values (for domain separation and to mitigate multi-target attacks) are treated and processed differently than the private ones. The $\mathbf{F}(\mathbf{PK}.\text{seed}, \mathbf{ADRS}, M_1)$ function of SLH-DSA, for example (responsible for the vast majority of function calls within the scheme, see Table 1), is supposed to map an n-byte input M_1 to an n-byte output under an $(n+32)$-byte domain separator (n-byte public seed $\mathbf{PK}$.seed and 32-byte address $\mathbf{ADRS}$), with n being 16, 24 or 32 depending on the targeted security level of $n \times 8$ bits. This separation is merely lost once the internal functions are mapped to concrete instances based on standardized hash functions like SHA-2 and SHA-3. In these instances, public and private inputs are simply concatenated and digested as one message, requiring a uniform protection of the entire computation circuit. A dedicated tweakable construction could potentially fit this task better.

Implementation Security. The implementation *security* argument refers to the fact that recent literature attributes very strong physical security against both side-channel and fault attacks to masked arithmetization-oriented (AO) primitives [9, 37, 49, 63, 85, 85, 92]. Even more importantly, these attributed resistances have been demonstrated to scale drastically with the underlying field size [55, 56, 71, 85], providing strong incentives to increase the cardinality of the underlying prime field as far as efficiency permits. For example, when integer arithmetic is readily available on a given implementation platform, it is typically advisable to scale the size of the prime close to, but slightly below, a small integer multiple of the single-precision operand size. On 32-bit platforms, for example, this has been achieved by selecting the 31-bit Mersenne prime $p = 2^{31} - 1$ which enables highly efficient modulo reduction [9].

Concretely, empirical results from [9] have demonstrated that a profiled side-channel attack on a tweakable block cipher implementation masked over $\mathbb{F}_2$ can succeed with more than 2 orders of magnitude fewer observations than an equivalent implementation masked at the same order over $\mathbb{F}_{2^{19}-1}$ [9, Fig. 7 vs. Fig. 8c]. This gap grows to almost 5 orders of magnitude (although based on extrapolation results) when considering an equivalent implementation masked over $\mathbb{F}_{2^{31}-1}$ [9, Fig. 7 vs. Fig. 8d]. Regarding active attacks, simulation results from [92] have shown that Statistical Ineffective Fault Attacks of Type-1 (SIFA-1) [42, 43] targeting circuits masked over $\mathbb{F}_2$ can succeed with few observations regardless of the masking order, e.g., between 10^2 and 10^3 fault attempts against any number of shares considering the adversary in [92, Fig. 5a]. In contrast, an equivalent circuit masked over $\mathbb{F}_{2^{19}-1}$ with 4 shares forces the same adversary to perform between 10^6 and 10^7 fault attempts for partial key recovery. Finally, the effectiveness of operation-dependent SIFA-2 attacks on masked gadgets typically relies on events occurring with probability $\frac{1}{\text{field size}}$ for collecting sensitive information, regardless of the number of shares [92]. For masking over $\mathbb{F}_2$ this

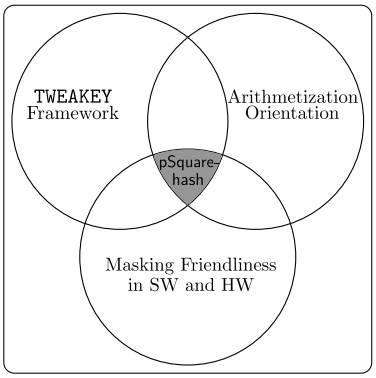


Figure 1. pSquare-hash's design space is the intersection of three paradigms.

naturally happens with probability $5 \cdot 10^{-1}$ while for masking over $\mathbb{F}_{2^{31}-1}$ this results in a probability as low as $4.7 \cdot 10^{-10}$, a difference of 9 orders of magnitude.

The Need for New Designs. When aiming to build physically secure and efficient implementations of stateless hash-based signature schemes, the underlying hash function matters. In particular, it matters beyond standard efficiency metrics. Based on the discussion above we conclude that the ideal hash function for our purposes would combine three different traits as illustrated in Figure 1. First, it should be tweakable to leverage the implementation efficiency argument made above. Second, it should be arithmetization-oriented, operating over a field of sufficient size, to leverage the implementation security argument made above. Third, it should be masking friendly in both software and hardware implementations. To the best of our knowledge, no current hash function falls into the intersection of our requirements, motivating the need for a new design. Creating such a new design can be approached from multiple angles. Turning standard masking-friendly lightweight permutations operating over binary fields (e.g., those of Keccak [22] or `Ascon` [45]) into tweakable constructions is feasible, but not trivial and likely requires tool-based cryptanalysis. In Section 6, we briefly discuss the alternative idea of using a smaller lightweight hash function without tweak and masking only those permutation calls in sponge or duplex mode that actually process private inputs. While this can improve over the standardized constructions, it requires consecutive permutation calls that cannot be parallelized, the unmasked calls are still presumably more expensive than a tweak schedule and it lacks elegance in hardware as both a protected and an unprotected circuit of the same logic function would be needed. Even more importantly, such instances could not leverage the implementation security argument above and would remain susceptible to low-noise side-channel attacks, transitional leakages in microcontrollers and most types of fault attacks. In contrast, efficient instances of established AO hash functions (e.g., Poseidon [61] or Poseidon2 [62]) could be utilized and made tweakable relatively easily (following the LED strategy of adding the tweakey only every few rounds [65]) in order

to leverage both the implementation efficiency and security argument. However, hardware efficiency and masking friendliness remain common bottlenecks for such instances. Our pSquare-hash design aims at balancing all three desired traits. In order to separate public and private inputs, we adopt the `TWEAKEY` framework [73] and adapt it to the hash function domain. The round function is based on a Feistel structure with squaring modulo $p = 2^{31} - 1$ as the only non-linear element. While the arithmetization orientation does not come for free and partially eats into the efficiency budget freed by the tweakability, the resulting security benefits are expected to significantly outweigh its cost (see empirical evidence listed above).

3 Modes of Operation

In this section we show how to construct a hash function and a compression function on top of a tweakable permutation TP (e.g., the tweakable pSquare-hash permutation H specified in Section 4), by introducing two modes of operation. The first mode of operation TS, is based on the sponge construction and transforms TP into a variable-input-length fixed-output-length tweakable hash function. We will show TS is secure insofar as it is indifferentiable from a random oracle, i.e., it can be replaced by a random oracle in most security proofs, provided the number of queries remains below a certain threshold. While the second mode, TC, transforms pSquare-hash into a fixed-input-length fixed-output-length tweakable compression function. Both of these modes are necessary for SPHINCS$^+$ and other hash-based signature schemes mentioned in Section 1.

Tweakable Permutations. Strictly seen, tweakable permutations are just block ciphers, where the key is not secret. More formally, a tweakable permutation over a finite field $\mathbb{F}_p$ is a function taking a tweak from $\mathbb{F}_p^\tau$ and a message that is an element from $\mathbb{F}_p^n$, where τ and n are predetermined positive natural numbers, and outputting an element from $\mathbb{F}_p^n$. Additionally, if the tweak is fixed, the resulting partial function from $\mathbb{F}_p^n$ to $\mathbb{F}_p^n$ is a permutation.

Definition 1 (Tweakable Permutation). *Let p be a prime power and let $n, \tau \geq 1$, then a tweakable permutation $\mathsf{TP} : \mathbb{F}_p^n \times \mathbb{F}_p^\tau \to \mathbb{F}_p^n$ is a function such that for each choice of $t \in \mathbb{F}_p^\tau$, $\mathsf{TP}(\cdot, t)$ is a permutation.*

If the underlying finite field is clear from context we denote by $\mathsf{tperm}(n, \tau)$ the space of tweakable permutations with permutation size n and tweak size τ.

For our security bounds in Section 3.4 we work with the *random tweakable permutation model* which we define next.

Definition 2 (Random Tweakable Permutation). *Let p be a prime power and let $n, \tau \geq 1$. We say that $\mathsf{TP} : \mathbb{F}_p^n \times \mathbb{F}_p^\tau \to \mathbb{F}_p^n$ is a random tweakable permutation if for every tweak $t \in \mathbb{F}_p^\tau$, $\mathsf{TP}(\cdot, t)$ is an independent and uniform sampled permutation of $\mathbb{F}_p^n$.*

A direct application of the previous remark about the similiarity of tweakable permutations and block ciphers is that choosing a random tweakeable permutation is equivalent to working in the ideal cipher model.

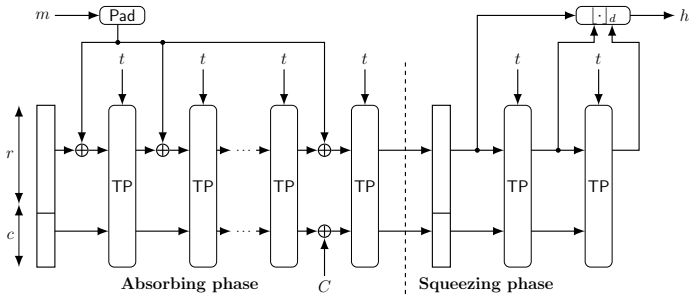


Figure 2. Tweakable sponge function. The symbol $\lfloor \cdot \rfloor_x$ denotes the truncation to the first x $\mathbb{F}_p$-elements.

3.1 The Tweakable Sponge Hash Function

Our first contribution is the tweakable sponge hash function, denoted by TS, which is obtained by instantiating a sponge hash function with a tweakable permutation.

Background: The Sponge Construction. The sponge hash function construction was first introduced by Bertoni et al. [20], and was later adopted as the SHA-3 hash function standard [97]. At a high level, it uses a cryptographic permutation P with n-bit state. It splits this state into two distinct parts: an outer part of size r, called the *rate*, and an inner part of size c, the *capacity*. To hash a message m, the sponge first *injectively pads* m to a string of size a multiple of r bits and subsequently cut into r-bit blocks: $m_0, m_1, \ldots, m_{\ell-1}$. Then, starting from an initial value $\mathtt{IV}$, these blocks are then XORed to the outer part of the state, interleaved by an evaluation of P on the whole state. Since the inner part is never touched by the message, the resulting state after absorbing each block remains "unpredictable". This phase is called *absorption*. Thirdly, after the last message block is absorbed, the d-bit digest is squeezed from the outer part of the state in r-bit blocks, again interleaved by evaluations of P. This last phase is called *squeezing*. The sponge construction remains secure if the number of evaluations of P is much lower than $2^{c/2}$. While the original sponge construction was defined over bits, it can be easily adapted to the setting of finite fields [75]. In this case, the sponge construction remains secure if the number of evaluations of P is much lower than $p^{c/2}$, for p being the prime power that defines the field.

The Tweakable Sponge Construction. The tweakable sponge hash function TS is obtained by instantiating the sponge construction just recalled with a tweakable permutation TP (denoted also by $\mathsf{TS}^{\mathsf{TP}}$). An illustration of such construction is given in Figure 2, while a formal description of TS is given in Algorithm 1 – see Appendix A. As in this paper we are going to deal with elements of finite fields (which are often large), TS uses block-length preserving padding [66, 82], that is, given an input message m and a rate r, we pad it as follows:

– if the length of m is not a multiple of the rate, we pad m with 10^*, since the expansion caused by padding *does not* result in an additional evaluation;
– if the length of m is a exactly a multiple of the rate, we do not pad, instead we make use of non-cryptographic permutations, that is, in the final absorption call, we sum in the capacity by a non-zero constant.

3.2 The Tweakable Compression Function

Next, we focus on tweakable compression functions. Compression functions take as input an element from $\mathbb{F}_p^n$ and output an element from $\mathbb{F}_p^d$ for $d < n$. Compression functions are of great interest [31,59,60,61,112], as they are crucial for instantiating a tree-hash function (besides Merkle-Damgård hash functions).

A tweakable compression function is just a compression function taking an additional input, namely a tweak. One can of course use TS as a compression function, however, in the use-case where only collision and preimage resistance properties are interesting, and not so much randomness per se (c.f. Section 3.3), then there are more efficient alternatives. One particular technique that is used extensively to create compression functions is to combine the truncation function with the feed-forward operation. A slightly more general approach is the tweakable compression function construction TC. Given a tweakable permutation $\mathsf{TP} \in \mathsf{tperm}(n, \tau)$ and an invertible matrix $M \in \mathbb{F}_p^{n \times n}$, then

$$(x, t) \in \mathbb{F}_p^n \times \mathbb{F}_p^\tau \mapsto \mathsf{TC}(x, t) \coloneqq \lfloor \mathsf{TP}(x, t) + M \times x \rfloor_d \in \mathbb{F}_p^d \,, \tag{2}$$

where $\lfloor \cdot \rfloor_d : \mathbb{F}_p^n \to \mathbb{F}_p^d$, for $1 \leq d < n$, denotes the truncation function that returns the first d $\mathbb{F}_p$-elements. As shown in [59], the multiplication with M does not affect the security in the case in which, both no constraint is imposed on the input x, and TP is an ideal tweakable permutation. For this reason, in this paper we simply fix $M = I$ (i.e., identity matrix).

3.3 Security: Indifferentiability and Various Kinds of Resistance

Next, we study the security properties of TS and TC. We start with indifferentiability, which is relevant for TS, and then move on to collision and preimage resistance properties, which are relevant for both constructions.

Indifferentiability. The indifferentiability framework was first introduced by Maurer et al. [87], and refined to the context of hash functions by Coron et al. [39]. In this work, we adapt this framework to the case of tweakable hash constructions. In the indifferentiability game, an adversary $\mathcal{A}$ is placed in either of two worlds: the real world, or the ideal world. In the real world, at the beginning of the game, a tweakable permutation TP is uniformly sampled from the set $\mathsf{tperm}(n, \tau)$. Then, $\mathcal{A}$ has access to the oracles $\mathsf{TS}^{\mathsf{TP}}$ and TP, called the left and right oracle respectively. In the ideal world, $\mathcal{A}$ is given access to two oracles with the same interface as before: the left oracle RO and the right oracle $\mathsf{Sim}^{\mathsf{RO}}$, where Sim is a simulator, that is a probabilistic Turing machine emulating TP.

At a high level, the goal of Sim is to behave like a random tweakable permutation, while at the same time ensuring that the corresponding tweakable sponge behaves like a random oracle.

The resources of $\mathcal{A}$ are limited, in particular, $\mathcal{A}$ may only make Q queries to its right oracle (TP or $\mathsf{Sim}^{\mathsf{RO}}$). Notice that in the real world, queries to the left oracle $\mathsf{TS}^{\mathsf{TP}}$ result in queries to TP, that is, the right oracle. Hence, in the ideal world, we compute queries to RO by the number of queries they would have required in the real world. Then, $\mathcal{A}$ interacts with whichever world it is placed, and outputs either 0 or 1 and we define its advantage with respect to Sim as:

$$\mathsf{Adv}^{\mathtt{if}}_{\mathsf{TS}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}\left[\mathsf{RO}, \mathsf{Sim}^{\mathsf{RO}}\right] \to 1\right] - \Pr\left[\mathcal{A}\left[\mathsf{TS}^{\mathsf{TP}}, \mathsf{TP}\right] \to 1\right]\right| .$$

We say that TS is indifferentiable from a random oracle if there exists a simulator Sim, that makes its advantage negligible in Q. Additionally, we denote by $Q_{\mathcal{T}}$ the maximum number of queries made with the same tweak. Note that it is always the case that $Q_{\mathcal{T}} \leq Q$.

Collision, Preimage, Second-Preimage and Collision Resistance. In the remainder of this section, we define collision resistance, (everywhere) preimage resistance, and (everywhere) second-preimage resistance [105].

Definition 3. *Let* $\mathsf{TX} \in \{\mathsf{TS}, \mathsf{TC}\}$ *be a construction relying on an inner tweakable permutation* TP*. Additionally, consider an adversary* $\mathcal{A}$ *making at most* Q *queries to* TP*. The collision resistance, (everywhere) preimage resistance, and (everywhere) second-preimage resistance of* TX *against an adversary* $\mathcal{A}$ *are defined respectively as*

$$\mathsf{Adv}^{col}_{\mathsf{TX}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{TX}} \to \left((m,t),(m',t')\right) \,:\, (m,t) \neq (m',t') \,\wedge\, \mathsf{TX}(m,t) = \mathsf{TX}(m',t')\right],$$

$$\mathsf{Adv}^{pre}_{\mathsf{TX}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{TX}}(z) \to (m,t) \,:\, \mathsf{TX}(m,t) = h\right],$$

$$\mathsf{Adv}^{sec}_{\mathsf{TX}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{TX}}(m,t) \to (m',t') \,:\, (m,t) \neq (m',t') \,\wedge\, \mathsf{TX}(m,t) = \mathsf{TX}(m',t')\right].$$

We remark that the notion of indifferentiability implies separately collision, preimage and second-preimage resistance up to the bounds that a random oracle would achieve [3, 4, Appendix B], that is, $\mathsf{Adv}^{\mathtt{pre}}_{\mathsf{TS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{if}}_{\mathsf{TS}}(\mathcal{A}) + \frac{Q}{p^d}$, $\mathsf{Adv}^{\mathtt{col}}_{\mathsf{TS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{if}}_{\mathsf{TS}}(\mathcal{A}) + \frac{Q^2}{2p^d}$, and $\mathsf{Adv}^{\mathtt{sec}}_{\mathsf{TS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{if}}_{\mathsf{TS}}(\mathcal{A}) + \frac{Q}{p^d}$, where Q denotes the maximum number of queries. Nevertheless, it is often possible to achieve better bounds by dedicated analysis. As we will see in the following section.

Multi-Target Collision Resistance with Random Sampling. The notion of multi-target collision resistance $\mathsf{Adv}^{\mathrm{sm-rtcr,k}}_{\mathsf{TX}}(\mathcal{A})$ is quite technical, so for the sake of brevity we refer the reader to [47, Definition 6]. At a high level, the security game of this notion works as follows: A challenger samples k values uniformly at random, alongside with a random salt, then it checks if any of these values satisfies the property `Prop` and outputs the last value that does satisfy `Prop` and gives it to the adversary. The adversary wins if it finds a second preimage for any of the sampled valid values by the challenger.

Table 2. Summary of security bounds for TP and TC. All bounds are given in $\log_p$-big $\mathcal{O}$ notation. A factor of $\log p$ is excluded.

Cons.	Indiff.	Collision	Preimage	Second-preimage
TS	$\frac{c}{2}$	$\frac{c}{2}$	$\min\left\{d, \max\left\{d-r, \frac{c}{2}\right\}\right\}$	$\min\left\{d, \frac{c}{2}\right\}$
TC	1	$\frac{d}{2}$	d	d

3.4 Security Statements

The different security guarantees for TS and TC with respect to indifferentiability, collision, first and second-preimage resistance are summarized in Table 2. As we prove in Appendix M, TS is indifferentiable, up to $\mathcal{O}\left(p^{\frac{c}{2}}\right)$ queries, and second-preimage resistant up to $\mathcal{O}\left(p^{d}\right)$ queries. On the other hand, TC enjoys first and second preimage security up to $\mathcal{O}\left(p^{d}\right)$ queries and collision resistance up to $\mathcal{O}\left(p^{d/2}\right)$, which is optimal.

In the remainder of the section, we assume that $\mathcal{A}$ is an adversary making at most Q unique permutation queries, and making at most $Q_{\mathcal{T}}$ queries with the same tweak. The following results (proven in Appendix M) hold:

Theorem 1. *Let* TS *be based on a random tweakable permutation* $\mathsf{TP} \in \mathsf{tperm}(n, \tau)$ *with capacity* c*. Then, there exists a simulator* Sim*, such that for any adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{if}_{\mathsf{TS}}(\mathcal{A}) \leq \frac{9QQ_{\mathcal{T}}}{p^c} + \frac{3Q}{p^c} + \frac{3\binom{Q}{2}}{p^n}\,.$$

with Q *denoting the total number of queries, and* $Q_{\mathcal{T}}$ *the maximum number of queries made with the same tweak.*

Our proof of Theorem 1, which can be found in Appendix M, ressembles that presented by Naito and Ohta [95]. However, it now includes the additional parameter $Q_{\mathcal{T}}$, which makes the bound finer (e.g., if tweak reuse is small, namely $Q_{\mathcal{T}} \ll Q$, our bound is *sharper*), and the analysis more subtle. To roughly explain the term $\frac{9QQ_{\mathcal{T}}}{p^c}$ in the bound, consider the following. In the sponge, the indifferentiability bound is dominated by a term of order Q^2/p^c, which accounts for collisions in the capacity created by an adversary in the indifferentiability game. (Remember that a collision in the capacity can be easily escalated to a full-state collision, which allows the adversary to easily distinguish the sponge from a random function.) In the case of TS, this is not the case, as a full-state collision caused by two messages (m, t) and (m', t') represent an advantage for the distinguisher only in the case in which $t = t'$.

Proposition 1. *Let* TS *be based on a random tweakable permutation* $\mathsf{TP} \in \mathsf{tperm}(n, \tau)$ *with capacity* c *and digest* d*. Then for any adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{col}_{\mathsf{TS}}(\mathcal{A}) \leq \mathsf{Adv}^{if}_{\mathsf{TS}}(\mathcal{A}) + \frac{\binom{Q}{2}}{p^d}\,, \tag{3}$$

$$\mathsf{Adv}^{pre}_{\mathsf{TS}}(\mathcal{A}) \leq \frac{8Q}{p^d} + \min\left\{\frac{4Q}{p^{n-r}} \cdot \left\lceil \frac{n}{r} \right\rceil, \frac{Q(Q-1)}{p^c}\right\}, \tag{4}$$

$$\mathsf{Adv}^{sec}_{\mathsf{TS}}(\mathcal{A}) \leq \mathsf{Adv}^{if}_{\mathsf{TS}}(\mathcal{A}) + \frac{Q}{p^d}, \tag{5}$$

The proofs of (3) and (5) follow from combining Theorem 1 with the results of [3,4] and the proof of (4) can be found in [83].

Regarding TC, the following results hold.

Proposition 2. *Let* TC *be based on a random tweakable permutation* $\mathsf{TP} \in \mathsf{tperm}(n,\tau)$ *with digest* d*. Then for any adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{col}_{\mathsf{TC}}(\mathcal{A}) \leq \frac{\binom{Q}{2}}{p^d - Q}, \tag{6}$$

$$\mathsf{Adv}^{sec}_{\mathsf{TC}}(\mathcal{A}) \leq \mathsf{Adv}^{pre}_{\mathsf{TC}}(\mathcal{A}) \leq \frac{Q}{p^d - Q}. \tag{7}$$

The bounds for TC are proven analogously as for Davies-Meyer mode in the ideal cipher model (see e.g. introductionary textbooks like [74, §7.3.1]), hence we omit them from the paper.

Finally, we re-propose the following result from [47, Appendix D] concerning multi-target collision security. Our analysis did not result in any concrete improvement that involves $Q_{\mathcal{T}}$.

Theorem 2. *Let* $\mathsf{TX} \in \{\mathsf{TS}, \mathsf{TC}\}$ *be based on a random tweakable permutation* $\mathsf{TP} \in \mathsf{tperm}(n,\tau)$ *with capacity* c *and digest* d*. Then for any adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathrm{sm-rtcr,k}}_{\mathsf{TX}}(\mathcal{A}) \leq \frac{2Qk}{p^c} + \frac{2Qk}{p^n} + \frac{4Qk}{p^d}.$$

4 Specification of pSquare-hash

In this section, we provide the details of the permutations used for pSquare-hash over $\mathbb{F}^8_{2^{31}-1}$, $\mathbb{F}^{12}_{2^{31}-1}$, $\mathbb{F}^{16}_{2^{31}-1}$ and $\mathbb{F}^{24}_{2^{31}-1}$. In the following, given $\ell \geq 1$, $I_{\ell\times\ell}$ and $0_{\ell\times\ell}$ denote the $\ell \times \ell$ identity matrix and the zero matrix respectively.

4.1 High-Level Structure

The tweakable pSquare-hash permutation H operates as

$$\mathsf{H} : \mathbb{F}_p^n \times \mathbb{F}_p^\tau \to \mathbb{F}_p^n \tag{8}$$

for $p = 2^{31} - 1$, $n \in \{8, 12, 16, 24\}$, and $\tau \in \{0\} \cup \{4, 5, 6, \ldots, 2 \cdot n\}$. For each $t \in \mathbb{F}_p^\tau$, $\mathsf{H}(\cdot, t)$ is invertible, that is, it is a permutation over $\mathbb{F}_p^n$. It is defined as

$$\begin{aligned}\mathsf{H}(x,t) = L_{n\times n} \times \big(t[R] + \mathsf{R}_{4\cdot R-1}\big) \circ \mathsf{R}_{4\cdot R-2} \circ \mathsf{R}_{4\cdot R-3} \circ \mathsf{R}_{4\cdot R-4} \circ \ldots \\ \ldots \circ (t[1] + \mathsf{R}_3) \circ \mathsf{R}_2 \circ \mathsf{R}_1 \circ \mathsf{R}_0\big(L_{n,n} \times (t[0] + x)\big),\end{aligned} \tag{9}$$

where

– $4 \cdot R$ is the number of rounds;
– $L_{n\times n} \in \mathbb{F}_p^{n\times n}$ is an invertible matrix defined in Section 4.2;
– $\mathsf{R}_i : \mathbb{F}_p^n \to \mathbb{F}_p^n$ is the invertible round function defined in Section 4.3;
– $t[0], t[1], t[2], \ldots, t[R] \in \mathbb{F}_p^n$ are the round-tweaks added every 4 rounds, and they are defined for $i \geq 0$ as
  - if $\tau \leq n$: $t[i] \coloneqq \begin{cases} 0, & \text{if } \tau = 0, \\ t'[i] \| \mathbf{0}^{n-\tau}, & \text{otherwise } (\tau \leq n), \end{cases}$
  - if $n < \tau \leq 2 \cdot n$: $t[2 \cdot i] \| t[2 \cdot i + 1] \coloneqq t'[i] \| \mathbf{0}^{2 \cdot n - \tau}$,

  where $\mathbf{0}^z \in \mathbb{F}_p^z$ is the zero element, $t'[0] \coloneqq t$, and the tweak schedule $t'[i+1] = \mathsf{T}\big(t'[i], i\big)$ with $\mathsf{T} : \mathbb{F}_p^\tau \times \mathbb{Z}_{\geq 0} \to \mathbb{F}_p^\tau$ is defined in Section 4.4.

Following small-pSquare [63] and mid-pSquare [9], we sometimes refer to $\mathsf{B}_i(\mathbf{x}) = \mathsf{R}_{4i+3} \circ \mathsf{R}_{4i+2} \circ \mathsf{R}_{4i+1} \circ \mathsf{R}_{4i}(\mathbf{x})$ as the i-th block of rounds B_i.

4.2 Input and Output Matrices

Our tweak permutations start and end with a matrix multiplication. The invertible matrices $L_{n\times n}$ for $n \in \{8, 12, 16, 24\}$ are defined as

$$L_{n\times n} \coloneqq \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \otimes I_{\frac{n}{2}\times\frac{n}{2}} = \begin{bmatrix} I_{\frac{n}{2}\times\frac{n}{2}} & I_{\frac{n}{2}\times\frac{n}{2}} \\ 2 \cdot I_{\frac{n}{2}\times\frac{n}{2}} & I_{\frac{n}{2}\times\frac{n}{2}} \end{bmatrix}. \tag{10}$$

4.3 Round Function R

Given n being a multiple of 4, the round functions R_i over $\mathbb{F}_p^n$ are a *slightly modified version* of the "Extended Generalized Feistel Networks" (EGFN) as defined in [16] by Berger et al., that is, a Type-II generalized Feistel network followed by a linear layer for increasing the diffusion.

The i-th round function $(y_0, y_1, \ldots, y_{n-1}) \coloneqq \mathsf{R}_i(x_0, x_1, \ldots, x_{n-1})$ is defined as

$$\begin{bmatrix} (y_0, y_1) \\ \vdots \\ \left(y_{\frac{n}{2}-2}, y_{\frac{n}{2}-1}\right) \\ \left(y_{\frac{n}{2}}, y_{\frac{n}{2}+1}\right) \\ \vdots \\ (y_{n-2}, y_{n-1}) \end{bmatrix} \coloneqq \left[\begin{array}{c|c} \mathsf{G}_i & I_{\frac{n}{2}\times\frac{n}{2}} \\ \hline I_{\frac{n}{2}\times\frac{n}{2}} & 0_{\frac{n}{2}\times\frac{n}{2}} \end{array}\right] \times \begin{bmatrix} (x_0, x_1) \\ \vdots \\ \left(x_{\frac{n}{2}-2}, x_{\frac{n}{2}-1}\right) \\ \left(x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\right) + \mathsf{F}_{i,0}\left(x_0, x_1\right) \\ \vdots \\ (x_{n-2}, x_{n-1}) + \mathsf{F}_{i,\frac{n}{4}}\left(x_{\frac{n}{2}-2}, x_{\frac{n}{2}-1}\right) \end{bmatrix}, \tag{11}$$

where

– G_i is defined as

$$C_{2\times 2} \coloneqq \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \mathsf{G}_i \coloneqq \begin{bmatrix} 0_{2\times2} & 0_{2\times2} & \ldots & 0_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & 0_{2\times2} & \ldots & 0_{2\times2} & C_{2\times2} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0_{2\times2} & 0_{2\times2} & \ldots & 0_{2\times2} & C_{2\times2} \\ 0_{2\times2} & I_{2\times2} & \ldots & I_{2\times2} & C_{2\times2} \end{bmatrix} \in \mathbb{F}_p^{\frac{n}{2}\times\frac{n}{2}}; \tag{12}$$

– $\mathsf{F}_{i,j} : \mathbb{F}_p^2 \to \mathbb{F}_p^2$ is defined as a 2-round Feistel, instantiated with $x \mapsto x^2$, in which the rotation of the inputs is replaced by a multiplication with a matrix $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, that is,

$$\mathsf{F}_{i,j}\begin{pmatrix} x \\ y \end{pmatrix} \coloneqq \begin{pmatrix} \left((y + c_{i,j})^2 + x + y + c'_{i,j}\right)^2 + (y + c_{i,j})^2 + x \\ \left((y + c_{i,j})^2 + x + y + c'_{i,j}\right)^2 + 2 \cdot (y + c_{i,j})^2 + 2 \cdot x + y + c'_{i,j} \end{pmatrix}, \tag{13}$$

where $c_{i,j}, c'_{i,j} \in \mathbb{F}_p$ are round constants.

We provide circuit representations of the round function in Figure 3.

Round Constants Generation. Round constants are generated in a similar manner as for mid-pSquare – details given in Appendix B.

4.4 Tweak Schedule

Let $\tau \in \{4, 5, 6, \ldots, 2 \cdot n\}$ be the size of the tweak, and let $t = (t_0, t_1, \ldots, t_{\tau-1}) \in \mathbb{F}_p^\tau$ be the tweak. Given $t'[0] \coloneqq t \in \mathbb{F}_p^\tau$, we define the i-th tweak $t[i] \in \mathbb{F}_p^n$ as in Section 4.1, where $t'[i+1] \coloneqq \mathsf{T}(t'[i], i)$ for the tweak schedule $\mathsf{T} : \mathbb{F}_p^\tau \times \mathbb{Z}_{\geq 0} \to \mathbb{F}_p^\tau$. T is analogous to a round of an SPN cipher defined as

$$\mathsf{T}(t, i) \coloneqq \left(2^{i+2} - 1, 0, 0, \ldots, 0\right) + \mathsf{T}_M \times \mathsf{T}_S(t), \tag{14}$$

where $2^{i+2} - 1$ is a round constant, and where

– the non-linear invertible function $\mathsf{T}_S : \mathbb{F}_p^\tau \to \mathbb{F}_p^\tau$ is defined as $\mathsf{T}_S(t_0, \ldots, t_{\tau-1}) \coloneqq \mathsf{T}'_S(t_0) \| \ldots \| \mathsf{T}'_S(t_{\tau-1})$, where

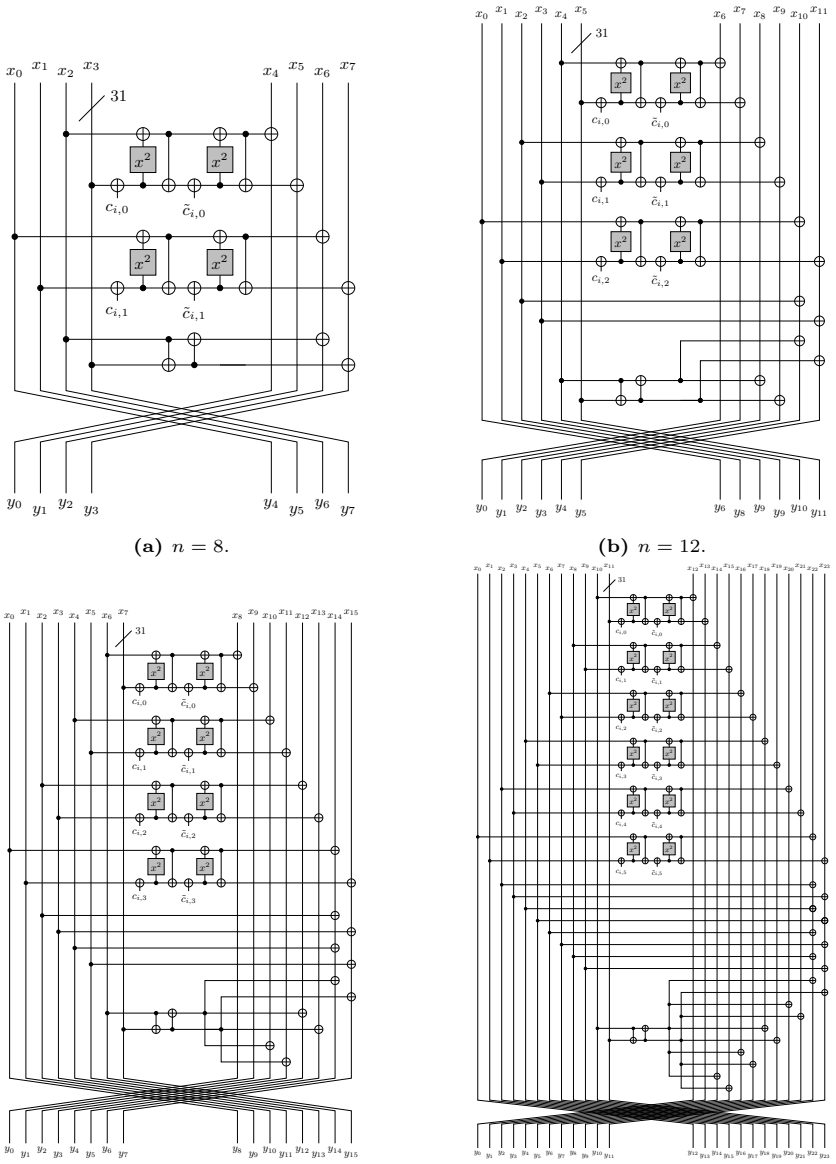
$$\mathsf{T}'_S(x) \coloneqq \Lambda^{-1}\left(\Lambda(x) \oplus \left(\Lambda(x) \gg 1\right) \oplus \underbrace{\Lambda\left(2^{30} - 1\right)}_{\text{constant}}\right), \tag{15}$$

where $\Lambda : \mathbb{F}_p \to \mathbb{F}_2^{31} \setminus \{\mathbf{1}\}$ is the invertible map that returns the bit-string corresponding to the input,[9] where $\mathbf{1}$ denotes the all 1 bit-string, $\oplus$ is the bitwise XOR, and $\gg$ is a bit-wise right shift;
– next, the entire state is multiplied by the $\tau \times \tau$ right circulant invertible (for $(\tau + 1) \mod p \neq 0$) matrix defined as $\mathsf{T}_M \coloneqq \mathrm{circ}(2, 1, 1, \ldots, 1)$.

4.5 Instances/Number of Rounds

In Table 3 we present pSquare-hash instances for standard arithmetization-oriented applications without a tweak. For sponge and compression mode we claim 124 bit of securty against *collision* and *(second-)preimage* attacks.

[9] Implicitly, we assume here that any input $x \in \mathbb{F}_p$ to Λ is represented as $0 \leq x \leq p-1$.



(a) $n = 8$. **(b)** $n = 12$. **(c)** $n = 16$. **(d)** $n = 24$.

Figure 3. pSquare-hash for $n \in \{8, 12, 16, 24\}$, where $\tilde{c}_{i,j} := c'_{i,j} - c_{i,j}$.

In Table 4 we present tweaked pSquare-hash instances for SPHINCS$^+$ signatures. We claim resistance of pSquare-hash against *Single-Function, Multi-Target,*

Table 3. Untweaked pSquare-hash instances (124 bits of security).

Mode	n	Rate	Capacity	Digest	Tweak Size	Blocks/Rounds
Sponge Hash Function	16	8	8	8	0	13/52
Sponge Hash Function	24	16	8	8	0	13/52
Compression	16	–	–	8	0	13/52

Table 4. Tweaked pSquare-hash instances for SPHINCS$^+$.

Mode	n	Rate	Capacity	Digest	Tweak Size	Blocks/Rounds	Application
Tweaked Sponge	24	16	8	8	12	15/60	SPHINCS$^+$ (124) Message hash
Tweaked Sponge	8	4	4	4	12	16/64	SPHINCS$^+$ (124) WOTS/FORS key gen. & WOTS chaining
Tweaked Compression	8	–	–	4	12	16/64	SPHINCS$^+$ (124) Merkle tree compression
Tweaked Sponge	24	16	8	12	14	15/60	SPHINCS$^+$ (186) Message hash
Tweaked Sponge	12	6	6	6	14	16/64	SPHINCS$^+$ (186) WOTS/FORS key gen. & WOTS chaining
Tweaked Compression	12	–	–	6	14	16/64	SPHINCS$^+$ (186) Merkle tree compression
Tweaked Sponge	24	16	8	16	16	15/60	SPHINCS$^+$ (248) Message hash
Tweaked Sponge	16	8	8	8	16	15/60	SPHINCS$^+$ (248) WOTS/FORS key gen. & WOTS chaining
Tweaked Compression	16	–	–	8	16	15/60	SPHINCS$^+$ (248) Merkle tree compression

Distinct-Tweaks (SM-DT) security notions[10] such as *Collision/Preimage Resistance* (SM-DT-TCR, SM-DT-PRE), *Decisional Second Preimage Resistance* (SM-DSPR) and *Undetectability* (SM-DT-UD). In particular, for a SPHINCS$^+$ instance over $\mathbb{F}_{2^{31}-1}$ let $\mathbb{F}_{2^{31}-1}^{d}$ be the digest space. Then we claim $31 \cdot d$ bit of classical security for pSquare-hash against SM-DT security notions.

4.6 Design Rationale

As the name suggests, pSquare-hash is inspired by small-pSquare [63] and mid-pSquare [9]. Still, the following crucial differences hold.

[10] For a precise definition of these notions and the security reduction of signature schemes, we refer the reader to [18, 69, 70]. As reference, [47, Table 1] summarizes classical/quantum bounds of these notions for random oracles.

Overall Construction. Similar to small-pSquare and mid-pSquare for $\tau \neq 0$, we decided to add the tweak every 4 rounds, which corresponds to the minimum number of rounds to get full diffusion. This allows us to provide a simple, efficient and elegant security argument against related-tweak statistical attacks.

We decided to replace the generalized Type-II Feistel Network [102,111] with a slightly modified version of the EGFNs defined in [16], since the latter one allows for much faster diffusion (4 rounds instead of $2 \cdot \log_2(n)$) at the cost of few extra additions per round, and it is simpler to scale for several values of n. With respect to the original EGFN defined in [16], our function F works over $\mathbb{F}_p^2$. In order to get full diffusion within 4 rounds, we introduced the 2×2 matrix $C_{2\times 2}$ as a replacement of the identity matrix in [16]. Formally:

Theorem 3 ([16, Theorem 7]). *An "Extended Generalized Feistel Network" (EGFN) over $\mathbb{F}_p^n$ with n even achieves full diffusion over 2 rounds if $n = 2$, and 4 rounds otherwise (i.e., $n \geq 4$).*

Corollary 1. *The slightly modified version of the EGFN over $(\mathbb{F}_p^2)^n$ for $n \in \{4, 6, 8, 12\}$ proposed in Section 4.3 achieves full diffusion over 4 rounds.*

The proof of this result is totally equivalent to the proof of [16, Theorem 7].

The Round Function. The round function R is similar to the one of mid-pSquare. We refer to [9, §2.3] for a detailed discussion about the design rationale of R. We limit ourselves to point out the following. The function R has degree 4 at the cost of two square operations. Roughly speaking, the combination of EGFN with such function R allows us to achieve degree 2^ζ within $\zeta/2$ rounds, at the cost of $(2 \cdot n/4) \cdot \zeta/2 = n \cdot \zeta/4$ square operations. Compared to a Type-II Feistel scheme instantiated with a square operation over $\mathbb{F}_p$ (which requires ζ rounds for achieving the same degree, for a cost of $(2 \cdot n/2) \cdot \zeta = n \cdot \zeta/2$), our design achieves higher degree for a much smaller cost.

The initial matrix $L_{n\times n}$ has the goal to spread the inner part of the sponge before the application of the round function, in such a way that the inputs of the non-linear function F depend on the inner part of the sponge as well. In this way, the attacker cannot skip the first round in some attacks as done e.g. in [12]. Besides, for some parameter sets, they allow us to construct a Gröbner basis for our scheme, allowing for a simpler security argument. Similar considerations hold for the final matrix multiplication.

The Tweak Schedule. The invertible tweak schedule aims to achieve full diffusion within 1 round, and to have a complex algebraic representation over $\mathbb{F}_p$.

The non-linear function $x \mapsto \Lambda^{-1}\Big(\Lambda(x) \oplus \big(\Lambda(x) \gg 1\big) \oplus \Lambda\big(2^{30} - 1\big)\Big)$ has been introduced in [9]. In there, the authors showed that T'_S has degree $p - 2$, and its interpolation polynomial is dense over $\mathbb{F}_p$ (practically tested for $p \in \{7, 31, 127\}$). Based on this, and due to the fact that T_M provides full diffusion within one round, the tweak schedule has a *complex algebraic description*, making it hard to exploit in related-tweak algebraic attacks. Further considerations are given in Appendix D.1.

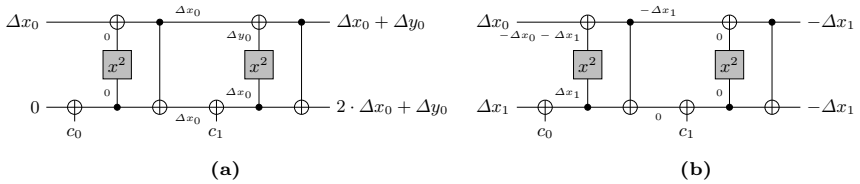


Figure 4. Differentials which activate exactly one square in the non-linear function.

5 Mathematical Security Analysis of pSquare-hash

In this section, we evaluate the security of pSquare-hash against standard statistical and algebraic attack vectors.

5.1 Statistical Cryptanalysis

Differential Attacks. In a differential attack [27], the idea is to study how the difference of two initial texts propagates through the symmetric primitive. As usual, the differential probability of having a certain output difference $\Delta_O \in \mathbb{F}_p^m$ given a particular input difference $\Delta_I \in \mathbb{F}_p^n$ through a function $\mathsf{H} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ is defined as $\text{Prob}(\Delta_I \to \Delta_O) = |\{x \in \mathbb{F}_p^n \mid \mathsf{H}(x + \Delta_I) - \mathsf{H}(x) = \Delta_O\}|/p^n$.

Case: $\tau = 0$. As usually done in the literature, in order to argue for the resistance of pSquare-hash against differential attacks for $\tau = 0$, we upper bound the probability of any differential characteresticProb$(\Delta_I \to \Delta_O)$ via $(\text{DP}_{\max})^s$, where $\text{DP}_{\max}$ is the maximum differential probability of F and s is the minimum number of active functions F. To compute this bound, note that:

- as shown in [9, §3.1.1] and Figure 4, the maximum differential probability of $\mathsf{F} : \mathbb{F}_p^2 \to \mathbb{F}_p^2$ is $p^{-1} \approx 2^{-31}$, which occurs in the case in which only one square function is active;
- we developed an automated search tool to count the number of active squares in differential trails starting from $\Delta_I \neq 0$ spanning r rounds. Details on our tool can be found in Appendix C. In Table 5 we list the maximal length of differential trails which activate between 1 and 8 squares found by our tool. According to our findings, at least 3 functions F are active over 4 consecutive rounds for each $n \in \{8, 12, 16, 24\}$. A concrete example of such differential is given by the first 4 rounds in Equation (16).

It follows that the maximum differential probability over 1 block (equivalently, 4 consecutive rounds) is at most $p^{-3} \approx 2^{-93}$.

We then compute the minimum number of blocks for guaranteeing security. Due to the clustering effect (i.e., the fact that several differential characteristics can be used together for setting up the attack) and due to the possibility of a Meet-in-the-Middle approach for setting up the attack, we claim that the scheme is secure if every differential characteristic has probability smaller than $2^{-2 \cdot \kappa}$ for

Table 5. Maximal lengths of found differential trails. Entries marked with $*$ are only for $n = 16$.

Active F		1	2	3	4	5	6	7	8
Rounds	$n = 8$	≤ 2	≤ 3	≤ 4	≤ 5				
	$n = 12$	≤ 2	≤ 3	≤ 4	≤ 5	≤ 5	≤ 6		
	$n = 16, 24$	≤ 2	≤ 3	≤ 4	≤ 5	≤ 5	$\leq 5^*$	$\leq 6^*$	$\leq 7^*$

Table 6. Maximal lengths of found truncated differential trails. Entries marked with $*$ are only for $n = 16$.

Active F		0	1	2	3	4	5	6	7	8
Rounds	$n = 8$	≤ 3	≤ 4	≤ 6	≤ 7	≤ 8				
	$n = 12$	≤ 4	≤ 5	≤ 6	≤ 6	≤ 7	≤ 8	≤ 9		
	$n = 16, 24$	≤ 4	≤ 5	≤ 6	≤ 6	≤ 7	≤ 8	$\leq 8^*$	$\leq 9^*$	$\leq 10^*$

a factor 2 (similar to what used in small-pSquare [63] and mid-pSquare [9]). As our target security level is $\kappa = 124$ for an untweaked ($\tau = 0$) sponge/compression function, 3 blocks of 4 rounds are sufficient to reach our goal, for a total of 12 rounds (note that $2^{-3 \cdot 93} = 2^{-279} = 2^{-2.25 \cdot 124} < 2^{-2 \cdot \kappa}$).

Case: $\tau \neq 0$. In the tweaked case the attacker is allowed to introduce differences within the tweaks in the related-tweakey model. As our target security levels for SPHINCS$^+$ are 124 ($n = 8$), 186 ($n = 12$) and 248 ($n = 16, 24$) bit for a tweaked sponge/compression function, 6 blocks of 4 rounds would be sufficient to reach our goals, for a total of 24 rounds (note that $2^{-6 \cdot 93} = 2^{-558} = 2^{-2.25 \cdot 248}$).

At the same time, following the argument proposed by LED's designers in [65] and already used for small-pSquare and mid-pSquare, the attacker can e.g. choose related tweaks such that only one out of two consecutive blocks B is active. As a result, in the case $\tau \leq n$, it is sufficient to double the number of blocks B just obtained to guarantee security, that is, 12 blocks or 48 rounds. If $n < \tau \leq 2 \cdot n$, two additional blocks are crucial for preventing attacks that exploit the larger size of the tweaks, for a total of 14 blocks or 56 rounds.

Truncated Differentials. Truncated differential cryptanalysis [76] generalizes differential cryptanalysis by considering differentials in which only a part of the difference is specified. With our tool we can also search for high-probability truncated trails. We note that our tool only searches for trails whose final differential still has at least one non-truncated element. For $n \in \{8, 12, 16, 24\}$, we list in Table 6 the maximal length of truncated differentials found by us, which activate between 0 and 8 functions F.

Case: $\tau = 0$. As full diffusion is achieved in 4 rounds, a truncated differential with probability 1 (equivalently, no active square) exists for up to 4 rounds. Example

trails for $n = 16$ are listed in Appendix C.1. As no truncated differential with probability higher than p^{-4} seems to exist for more than 7 rounds, we conjecture that $2 \cdot 8 = 16$ rounds guarantees security against truncated differentials.

At the same time, we emphasize that our tool is not able to detect truncated differentials with a *predictable pattern* (equivalently, linear relations) in the final truncated differential. Let us provide an example for $n = 16$ (analogous for $n = 24$). Let $C \equiv C_{2\times 2} \in \mathbb{F}_p^{2\times 2}$ as in Section 4.3, and let $\delta, \delta_1^*, \delta_2^* \in \mathbb{F}_p^2 \setminus \{0\}$, where δ_1^* denotes a truncated differential with $\delta_1^* \xrightarrow[\text{prob. } 1/p^2]{\mathsf{F}} -\delta$, then

$$
\begin{aligned}
& (0,0,0,\delta, \mid 0, -C\delta, -C\delta, -C\delta) && \xrightarrow[\text{prob. } 1]{\mathsf{R}} && (\delta_1^*, 0, 0, 0 \mid 0, 0, 0, \delta) \\
\xrightarrow[\text{prob. } \frac{1}{p^2}]{\mathsf{R}} \quad & (0,0,0,0 \mid \delta_1^*, 0, 0, 0) && \xrightarrow[\text{prob. } 1]{\mathsf{R}} && (\delta_1^*, 0, 0, 0 \mid 0, 0, 0, 0) \\
\xrightarrow[\text{prob. } 1]{\mathsf{R}} \quad & (0,0,0,\delta_2^* \mid \delta_1^*, 0, 0, 0) && \xrightarrow[\text{prob. } 1]{\mathsf{R}} && (*, C\delta_2^*, C\delta_2^*, C\delta_2^* \mid 0, 0, 0, \delta_2^*) \\
\xrightarrow[\text{prob. } 1]{\mathsf{R}} \quad & (*,*,*,* \mid *, C\delta_2^*, C\delta_2^*, C\delta_2^*). && && \qquad (16)
\end{aligned}
$$

Having said this, we conjecture that it is not possible to extend the truncated differentials found in Table 6 for more than 4 rounds by exploiting linear relations existing between the input entries and between the output ones. Based on this, we claim that $2 \cdot 8 + 4 = 20$ rounds (equivalently, 5 blocks) are sufficient for ensuring security against truncated differential attacks.

Case: $\tau \neq 0$. For $n = 8/12/16/24$ no truncated differential with probability higher than $p^{-4}/p^{-6}/p^{-8}$ seems to exist for more than 8/9/10 rounds respectively. So, based on our previous argument we require at least $2 \cdot 12 = 24$ rounds for security levels 124/186/248 bits respectively. For related tweaks, we recall that the tweak schedule is defined as an SPN scheme that achieves full diffusion within 1-round. For this reason, we expect that the attacker is not able to exploit truncated differentials of the tweak schedule for more than $\lceil \tau/n \rceil$ block(s). Hence, we conjecture that $2 \cdot \lceil \tau/n \rceil$ extra blocks are sufficient to achieve security against related-tweak truncated differential attacks. So, in summary we conjecture that $6 + 2 \cdot \lceil \tau/n \rceil \in \{8, 10\}$ blocks (equivalently, $24 + 8 \cdot \lceil \tau/n \rceil \in \{32, 40\}$ rounds) are sufficient to reach our security target of 124/186/248 bits respectively.

Other Statistical Attacks. The security against other statistical attacks (including linear attacks [6, 86], impossible differentials attacks [25, 26], and more) is proposed in Appendix D.2. We highlight that the previously given number of rounds is sufficient to guarantee security against these attacks as well.

5.2 Algebraic Cryptanalysis: Degree and Density of the Polynomial Representation of pSquare-hash

In order to claim security against algebraic attacks, we first discuss the degree and the density of the polynomial representation of r-round pSquare-hash.

Growth of the Degree. It is straightforward to verify that both the Feistel-based round function R and its inverse R^{-1} exhibit algebraic degree 4. Because of the Feistel design, only one half of the state is processed by the internal function in each round, while the other half bypasses it, thus preserving its degree. As a result, when considering r successive rounds of R (or R^{-1}), the lowest algebraic degree of the polynomial evolves as 4^{r-1} for $r \geq 1$. This leads to the conclusion that once $r \geq 17$, the minimal degree among these polynomials meets or exceeds the theoretical maximum for a single variable, i.e., $4^{r-1} \geq p - 1 = 2^{31} - 2$.

The smallest number of rounds after which at least one *complete* monomial (i.e., a monomial of the form $x^e = \prod_{i=0}^{n-1} x_i^{e_i}$ with $e_i > 0$ for each $0 \leq i \leq n-1$) appears is given by $4^{r-1} \geq n \cdot (p-1) = n \cdot (2^{31} - 2)$, which implies $r = 1 + \log_4(n \cdot (2^{31} - 2)) \approx 18$ for $8 \leq n \leq 24$. We emphasize that 18 is the *minimum* number of rounds necessary to reach maximum degree, but it does not guarantee that the maximum degree is reached. In particular, as shown by Beyne and Verbauwhede [24], the growth of the degree slows down when the degree is close to its maximum. We conducted an experimental analysis of the algebraic degree growth of pSquare-hash, and observed very similar behaviors in all cases. The corresponding results are shown in Figures 5a to 5d, while full details are deferred to Appendix E.[11] Based on this, we claim that 24 rounds are sufficient for the degree to be (almost) maximum, a sufficient condition for guaranteeing algebraic security when working with hash function constructions.

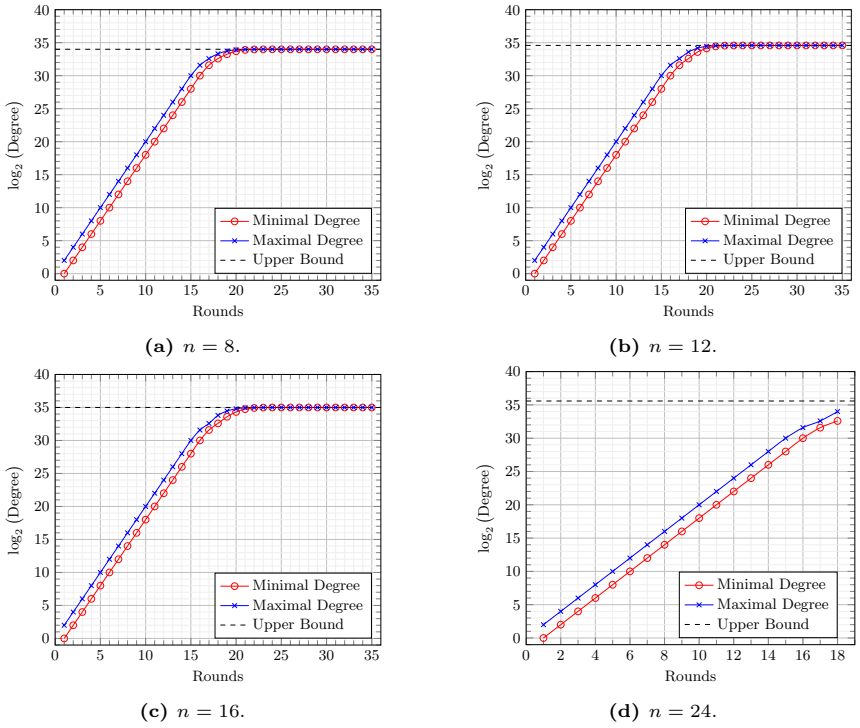
Density of the Polynomial Representation. Apart from the growth in algebraic degree, another key aspect in resisting algebraic attacks is the density of the polynomial representation.

Assumption 1. We say that the interpolation polynomial of pSquare-hash is dense if it contains $\frac{p-1}{p} \cdot p^n$ monomials.

Due to the large number n of variables, empirically counting the number of monomials becomes quickly intractable after a few rounds. For this reason, we focused on toy versions of pSquare-hash defined over $\mathbb{F}_p^{\tilde{n}}$, with $\tilde{n} - 4$ variables fixed, where $\tilde{n} = 8, 16$, for smaller primes p with $7 \leq p \leq 31$. (Here we exploit the simple scalability of our design, something that it is hard to achieve in a generalized Type-II Feistel [111], since each m required a dedicated shuffle, with a different diffusion propagation.) Our results are displayed in Appendix F Figures 7 and 8. By extrapolation (see Figures 7f and 8d), we estimate that 19 rounds are sufficient for the polynomial to be dense.

Interpolation Attack (and More). Based on the previous analysis, we claim that 12 blocks for $\tau = 0$ and $12 + \lceil \tau/n \rceil + 1 \in \{14, 15\}$ blocks otherwise (respectively, $4 \cdot 12 = 48$ and $4 \cdot (12 + \lceil \tau/n \rceil + 1) \in \{56, 60\}$ rounds) are sufficient to prevent algebraic attacks such as the interpolation attack [72]. We recall that the interpolation attack aims to re-construct the polynomial that defines the attacked

[11] For $n = 24$, we attempted to perform the same computations but found them prohibitively expensive in terms of memory consumption. So far, we have only been able to obtain results for up to 18 rounds.



(a) $n = 8$. **(b)** $n = 12$. **(c)** $n = 16$. **(d)** $n = 24$.

Figure 5. Degree growth of the pSquare-hash permutation.

scheme, and exploiting it to break the scheme (note that a Meet-in-the-Middle version is also feasible). Similar conclusion holds for the higher-order differential attack and the linearization attack as well, as explained in Appendix G. In particular, we emphasize that we do **not** make claims about the security of pSquare-hash against *zero-sum partitions* [30] – see Appendix G.3 for details.

5.3 Gröbner Basis Attacks (and More)

Background. Prime-field designs are often susceptible to polynomial system solving-based attacks [1, 10, 11, 34, 109, 116]. A general toolkit for solving fully determined polynomial systems are Gröbner Bases (GB), for an introduction to the theory we refer the reader to [40]. If $\mathsf{G} \subset \mathbb{F}_p[x_0, \ldots, x_{n-1}]$ is a fully determined Gröbner basis (with respect to some term order $>$), then one construct a univariate polynomial $\mathsf{U} \in \mathbb{F}_p[x_{n-1}]$ via G such that the roots of U are exactly the x_{n-1}-coordinates of all roots of G in $\mathbb{F}_p^n$. Moreover, via G one can accurately predict the degree of U. Henceforth, the complexity of solving a polynomial system $\mathsf{F} \subset \mathbb{F}_p[x_0, \ldots, x_{n-1}]$ depends on the complexities of finding some Gröbner basis G, the construction of a univariate polynomial $\mathsf{U} \in \mathbb{F}_p[x_{n-1}]$, and finally the factorization of U.

Table 7. Problems for pSquare-hash Gröbner basis attacks.

Compression Mode	Sponge Mode	
Preimage	Preimage	CICO
$\mathsf{H}\left(\mathbf{x}_{in}\right) + \mathbf{x}_{in} = \begin{pmatrix} \mathbf{h} \\ \mathbf{x}_{out} \end{pmatrix}$	$\mathsf{H}\begin{pmatrix} \mathbf{x}_{in} \\ \mathtt{IV} \end{pmatrix} = \begin{pmatrix} \mathbf{h} \\ \mathbf{x}_{out} \end{pmatrix}$	$\mathsf{H}\begin{pmatrix} \mathbf{x}_{in} \\ \mathtt{IV}_{in} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{out} \\ \mathtt{IV}_{out} \end{pmatrix}$

GB Attacks on pSquare-hash. Typical polynomial models in cryptographic settings are high-degree systems, which work only with the inputs/outputs of the function, or low-degree models, which introduce intermediate state variables after every round. For security of pSquare-hash we have to analyze the preimage problem for compression and sponge mode as well as the Constraint-Input Constraint-Output (CICO) problem for sponge mode, see Table 7 where $\mathsf{H} : \mathbb{F}_p^n \to \mathbb{F}_p^d$ denotes a pSquare-hash instance in the respective mode. The round level models for pSquare-hash consist of affine and quartic polynomials, which can be transformed into affine and structured quadratic polynomials via a preprocessing step. Henceforth, resistance of pSquare-hash against GB attacks reduces to structured cases of the multivariate quadratic problem over $\mathbb{F}_p$.

GB cryptanalysis of pSquare-hash is split into two cases:

(a) fully determined systems (sponge mode for $n = 8, 12, 16$),
(b) underdetermined systems (compression mode for $n = 8, 12, 16$ and sponge mode for $n = 24$).

For fully determined instances we can construct a quadratic GB with respect to the *degree reverse lexicographic* (DRL) term order. Thus, we can accurately predict the degree of the univariate polynomial U in the *lexicographic* (LEX) basis as $d = \deg(\mathsf{U}) = 2^{\frac{R \cdot n}{2}}$, where R denotes the total number of rounds of pSquare-hash. For factorization of U, one first computes $\mathsf{U}' = \gcd\left(\mathsf{U}(x_{n-1}), x_{n-1}^p - x_{n-1}\right)$, which restricts all roots of U' to $\mathbb{F}_p$. The GCD complexity is well approximated by $\mathcal{O}\left(d \cdot \log_2(d)^2 \cdot \log_2(\log_2(d))\right)$. We claim that these instances resist Gröbner basis attacks if d exceeds the target security level. So, $\frac{R \cdot n}{2} > \frac{n}{2} \cdot 31$ implies that $R > 31$ many rounds are sufficient for all instances in (a).

For underdetermined instances the situation is more subtle since we can arbitrarily guess some variables. While we could again identify guessing strategies which produce quadratic DRL Gröbner bases, this is in general not the case. Hence, we base the security on the hardness of computing a DRL Gröbner basis. Therefore, we performed extensive Gröbner basis experiments for guessing strategies which linearize the most number of squares. We observed, as one would expect, that the empirical complexity scales with the number of rounds R. The complexity of DRL Gröbner basis computations with the F4 algorithm [50] scales with $\mathcal{O}\left(\binom{n_{\text{vars}} + d_{\text{F4}}}{d_{\text{F4}}}^{\omega}\right)$, with n_{vars} the number of variables, d_{F4} F4's maximal internal working degree and $2 \leq \omega < 2.371552$ [115]. For pSquare-hash we have $n_{\text{vars}} \approx \frac{R \cdot n}{2}$. Based on our experiments, we formulate the conservative conjecture

Table 8. Cortex-M4 cycle count comparison of (masked) C-language software implementations of compression (SHA-2), permutation (Keccak, `SKINNY-Hash`, `Ascon`, Poseidon2) and tweakable permutation (pSquare-hash) functions with d shares.

Function	State [bits]	Tweak [bits]	$d=1$ [cyc]	$d=2$ [cyc]	$d=4$ [cyc]	$d=8$ [cyc]	$d=16$ [cyc]	$d=32$ [cyc]
SHA-2 compre.	512/256	0	2479	-	-	-	-	-
	1024/512		10477	-	-	-	-	-
Keccak-f	400	0	7848	29129	103902	256174	1722632	6101284
	800	0	7615	29269	109283	274523	1816776	6747358
	1600	0	14509	68679	230225	628093	4645857	17135350
Keccak-p[1600, 12]	1600	0	7260	32346	115123	314051	2322935	8567687
`SKINNY-Hash` F	256	0	16823	42689	151602	447288	1993672	6124556
	384		31324	76209	267336	784285	3482114	10719210
`Ascon-p`	320	0	1849	8435	36342	84000	544783	1881520
Poseidon2$^{\pi}$	496	0	12137	33563	131091	614599	2124731	8118574
	744		21481	56034	207270	941396	3242503	12553241
pSquare-hash	248	0	6250	21807	59590	302659	1048999	3822163
		248	9714	28702	70158	351518	1212788	4406129
		496	14999	34396	78377	379197	1298042	4709231
	372	0	10547	32531	87979	463300	1627758	5761577
		372	16714	43363	107547	539427	1883004	6655533
		744	23198	51918	119957	587312	1985002	7104064
	496	0	14574	45775	121184	626974	2164314	7732533
		496	22671	59180	145671	741949	2483494	9057140
		992	30757	69372	161479	783433	2623574	9664507
	744	0	24901	70002	187528	943518	3238392	11576098
		744	37406	91289	225124	1106619	3763104	13371788
		1488	49952	106332	251934	1175350	3988149	14530910

$d_{\text{F4}} \approx 4 \cdot \lfloor \frac{R}{4} \rfloor$, i.e. every block of 4 rounds increases F4's working degree by 4. According to our estimations with $\omega = 2$, $R = 32$ rounds are again sufficient for the complexity of pSquare-hash Gröbner basis computations to exceed the target security levels. Due to the page limit, details and experimental results of our Gröbner basis analysis are reported in Appendix G.5.

Rebound Attacks. In the case of arithmetization-oriented primitives, finding a Gröbner basis is crucial for finding solutions for the inbound phase of rebound attacks [80, 89]. Due to the page limit, we discuss the security of our design against the rebound attack in Appendix H.

6 Cost & Performance

In this section we evaluate the cost and performance of pSquare-hash implementations in different settings, including masked hashing and masked signatures. A detailed analysis of plain and proof system performance of pSquare-hash based on `Plonky3` [103] is provided in Appendices I and J, including a discussion of the costs in further proof systems as well as a practical evaluation using STARKs.

6.1 Masked Hashing in Software and Hardware

We begin by comparing the performance of our pSquare-hash tweakable permutations to the underlying compression functions of SHA-2 and permutations of SHA-3, `SKINNY-Hash`, `Ascon-Hash`, and Poseidon2. For Poseidon2, we have generated an efficient instance using the Mersenne prime $p = 2^{31} - 1$ instead of the more costly Babybear prime, by following the respective instantiation guidelines of the authors [62]. In a first benchmark we compare the cycle cost of constant-time 100% C-language implementations of the compression, permutation and tweakable permutation functions on a low-cost ARM Cortex-M4 32-bit microcontroller. In particular, we use the STM32F415RGT6 device which is part of the STM32F target for the ChipWhisperer CW308 UFO board. The binaries are compiled using Keil MDK Community Edition 5.42.0.0, compiler version V6.23, with the `-O3` and `-flto` flags to optimize for speed. All cycle counts are measured on the device itself using ST-LINK in-circuit debugger. The results are presented in Table 8. Since physical security is a central goal of this work, we compare not only unprotected ($d = 1$), but also (higher-order) masked implementations with up to 32 shares ($2 \leq d \leq 32$). The SHA-2 compression functions are excluded from the masked comparison, because both SHA-256 and SHA-512 are well known to be comparatively ill-suited for the secure and efficient application of the masking countermeasure due to their mix of Boolean and arithmetic operations. Existing literature on protected SLH-DSA implementations via the masking countermeasure generally concentrates on the SHAKE instances [106]. The masked implementations of the Keccak-f[1600] permutation are derived from source code published in conjunction with [32]. Corresponding implementations of smaller Keccak permutations (i.e., f[400] and f[800]) and reduced-round versions like those of TurboSHAKE [19] (i.e., p[1600, 12]) are adapted from the full version. The `Ascon` implementation is also derived from that source code. The SHA-256 and SHA-512 implementations are taken from the *slhdsa-c*[12] project. The masked pSquare-hash, Poseidon2 and `SKINNY-Hash` implementations reuse publicly available source code from [9]. For Poseidon2, the concrete construction of the masked x^5 power map was proven secure and trivially composable at any order in [37]. pSquare-hash constitutes the only tweakable permutation in the comparison. Since the tweak is considered public information, the tweak schedule can remain completely unmasked.

Unsurprisingly, it can be observed that the cost of the individual compression or (masked) permutation functions correlates with their underlying state size. Since functions with larger input sizes may digest data at a faster rate while maintaining the same security level, we require additional context to judge the efficiency and suitability of the functions for use in stateless hash-based signature schemes. This context is provided in Table 9. As mentioned earlier and shown in Table 1, SLH-DSA's $\mathbf{F}(\cdot)$ function is responsible for the vast majority of hash function calls within the scheme. It is used as a chaining function in the underlying one-time signature scheme as well as for multiple (sub-)key generation

[12] https://github.com/pq-code-package/slhdsa-c

Table 9. Minimum number of sub-function calls and Cortex-M4 cycles (based on Tab. 8) when instantiating (masked) SLH-DSA's $\mathbf{F}(\cdot)$ function via different hashes.

Security [bits]	Hash/XOF	Min. # of Calls	Min. # of Mask. Calls	Min. # of Cycl. ($d = 2$)	Min. # of Cycl. ($d = 32$)
124 to 128	SHA-256	2	1	-	-
	SHAKE256	1	1	68679	17135350
	`Ascon-Hash`	9	3	36399	5655654
	`SKINNY-tk3-Hash`	4	1	170181	10813182
	Poseidon2 ($t = 16$, compr.)	1	1	33563	8118574
	pSquare-hash ($n = 8$, $\tau = 12$)	1	1	**32886**	**4631837**
186 to 192	SHA-256	2	1	-	-
	SHAKE256	1	1	68679	17135350
	`Ascon-Hash`	12	5	55118	9420543
	`SKINNY-tk3-Hash`	6	3	322599	32251602
	Poseidon2 ($t = 24$, compr.)	1	1	56034	12553241
	pSquare-hash ($n = 12$, $\tau = 14$)	1	1	**49854**	**6980291**
248 to 256	SHA-256	2	1	-	-
	SHAKE256	1	1	68679	17135350
	`Ascon-Hash`	15	7	73837	13185432
	`SKINNY-tk3-Hash`	7	3	353923	32251602
	Poseidon2 ($t = 24$, compr.)	1	1	**56034**	12553241
	pSquare-hash ($n = 16$, $\tau = 16$)	1	1	59180	**9057140**

tasks. When optimizing the overall scheme's performance, this function is the most critical. Table 9 lists the minimum number of total calls to the underlying sub-functions (i.e., either compression, permutation or tweakable permutation) required per $\mathbf{F}(\cdot)$ function call when instantiated with different hashes, as well as the minimum number of calls that need to be masked even when optimally re-ordering the public and private inputs. These values are then provided for all three security levels targeted by the scheme [101]. It can be observed that the only primitives requiring a single sub-function call at all security levels are SHAKE256, Poseidon2 (in compression mode) and pSquare-hash. Additionally, based on these lower limits of sub-function calls and the performance numbers given in Table 8, we estimate the minimum number of Cortex-M4 cycles needed to mask $\mathbf{F}(\cdot)$ function calls at both $d = 2$ and $d = 32$ shares. Both our custom Mersenne-based Poseidon2 version and pSquare-hash outperform the SHAKE256-based implementation at any order. Asymptotically (for large d), they would even outperform corresponding TurboSHAKE variants at the lower security levels. For the other hash functions, multiple calls to the underlying compression/permutation are required, some of which need to be masked as they are processing secret inputs. In software this can be solved by consecutively executing masked and unmasked calls inside their sponge, duplex or Merkle-Damgård construction (instead of relying on a tweakable hash). Yet, these calls cannot be parallelized, they require constant re-masking of the state, and in hardware instantiating both unmasked and masked circuits for the same function only makes sense if the unmasked ones are significantly faster or if power consumption is the bottleneck.

A corresponding comparison of round-based (masked) hardware implementations is given in Tables 22 and 23 of Appendix K. The implementation properties

Table 10. Cortex-M4 cycle count comparison of (masked) C-language software implementations of SLH-DSA-SHA2, SLH-DSA-SHAKE and SPHINCS$^+$-pSquare with d shares. Results given in Mcycles.

Instance	$d=1$			$d=2$			$d=4$		
	Gen	Sign	Vrfy	Gen	Sign	Vrfy	Gen	Sign	Vrfy
SLH-DSA-SHA2-128f	12.4	291.0	17.3	-	-	-	-	-	-
SLH-DSA-SHA2-256f	50.3	1158.2	31.0	-	-	-	-	-	-
SLH-DSA-SHAKE-128f	91.4	2141.4	132.1	318.8	7425.4	458.7	1048.8	24489.2	1513.7
SLH-DSA-SHAKE-256f	353.4	7181.1	190.8	1225.4	24662.9	661.0	4041.5	81117.9	3820.8
SPHINCS$^+$-pSquare-124f	65.3	1527.8	99.4	171.6	4003.4	238.0	426.0	9802.9	575.3
SPHINCS$^+$-pSquare-248f	381.3	7739.0	218.8	1138.0	22815.6	614.5	2762.8	55391.8	1491.8

are evaluated in the NanGate 15 nm Open Cell Library using Synopsys Design Compiler Version O-2018.06-SP4. All of our implementations have been verified to provide glitch- and transition-robust probing security using the automated verification tools MATCHI [36] and/or PROLEAD [93]. The tradeoffs between speed and size are flexibly adjustable within certain limits. Hence, direct comparisons are not always trivial. However, round-based Poseidon2 clearly performs significantly worse than other competitors in area, power, critical path and randomness requirements, due to its use of two different round functions, one with full and one with partial non-linear layer, as well as the larger power map, requiring both, masked multiplication and squaring gadgets.

6.2 Masked Signing in Software

Finally, we compare the software performance results of full standardized SLH-DSA instances (focusing on the f variants) to our novel SPHINCS$^+$ instances in Table 10, based on the same ARM Cortex-M4 microcontroller and evaluation methodology described above. In Table 24 of Appendix L, we provide a simplified sensitivity analysis of SLH-DSA's internal functions to both side-channel and fault injection attacks on the confidentiality and integrity of the scheme. While for confidentiality alone, only the $\mathbf{F}$, $\mathbf{PRF}$, and $\mathbf{PRF}_{msg}$ functions in key and signature generation would need to be protected (no confidential data processed in verification), achieving integrity additionally requires protection of $\mathbf{H}$, $\mathbf{T}_l$ and to a lesser extent also $\mathbf{H}_{msg}$, even in verification. Therefore, the numbers given in Table 10 refer to implementations that mask all functions, yet only the sensitive paths. Overall, the cycle counts are rather high, indicating that protected SLH-DSA / SPHINCS$^+$ implementations on low cost microcontrollers are unrealistic without hardware acceleration. Yet, we strive for a relative comparison here. Our pSquare-hash variants of SPHINCS$^+$ generally outperform the protected SHAKE instances, and sometimes even the unprotected ones. As detailed in Section 2, this performance advantage comes on top of significant physical security benefits originating from the chosen algebraic setting. Overall, we conclude that pSquare-hash is very suitable for the efficient instantiation of physically secure hash-based signature schemes, warranting further evaluations.

Acknowledgments. 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. Thorben Moos is a postdoctoral researcher of the Belgian Fund for Scientific Research (F.R.S.-FNRS).

References

1. Albrecht, M.R., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 371–397. Springer, Cham, Switzerland, Kobe, Japan (Dec 8–12, 2019). https://doi.org/10.1007/978-3-030-34618-8_13
2. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Transactions on Symmetric Cryptology **2020**(3), 1–45 (2020). https://doi.org/10.13154/tosc.v2020.i3.1-45
3. Andreeva, E., Mennink, B., Preneel, B.: Security reductions of the second round SHA-3 candidates. Cryptology ePrint Archive, Report 2010/381 (2010), https://eprint.iacr.org/2010/381
4. Andreeva, E., Mennink, B., Preneel, B.: Security reductions of the second round SHA-3 candidates. In: Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.) ISC 2010: 13th International Conference on Information Security. Lecture Notes in Computer Science, vol. 6531, pp. 39–53. Springer Berlin Heidelberg, Germany, Boca Raton, FL, USA (Oct 25–28, 2011). https://doi.org/10.1007/978-3-642-18178-8_5
5. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F.X., van Vredendaal, C.: Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 58–79 (2023). https://doi.org/10.46586/tches.v2023.i4.58-79
6. Baignères, T., Stern, J., Vaudenay, S.: Linear cryptanalysis of non binary ciphers. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) SAC 2007: 14th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4876, pp. 184–211. Springer Berlin Heidelberg, Germany, Ottawa, Canada (Aug 16–17, 2007). https://doi.org/10.1007/978-3-540-77360-3_13
7. Bak, A.: A practical distinguisher on the full Skyscraper permutation. Cryptology ePrint Archive, Report 2025/102 (2025), https://eprint.iacr.org/2025/102
8. Bak, A., Jazeron, G., Galissant, P., Perrin, L.: Attacking split-and-lookup-based primitives using probabilistic polynomial system solving applications to round-reduced Monolith and full-round Skyscraper. IACR Transactions on Symmetric Cryptology **2025**(3), 337–367 (2025). https://doi.org/10.46586/tosc.v2025.i3.337-367
9. Balon, B., Grassi, L., Méaux, P., Moos, T., Standaert, F.X., Steiner, M.J.: midpSquare: Leveraging the strong side-channel security of prime-field masking in

software. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(4), 486–519 (2025). https://doi.org/10.46586/tches.v2025.i4.486-519

10. Bariant, A., Boeuf, A., Briaud, P., Hostettler, M., Øygarden, M., Raddum, H.: Improved resultant attack against arithmetization-oriented primitives. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025, Part V. Lecture Notes in Computer Science, vol. 16004, pp. 335–367. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2025). https://doi.org/10.1007/978-3-032-01901-1_11
11. Bariant, A., Boeuf, A., Lemoine, A., Ayala, I.M., Øygarden, M., Perrin, L., Raddum, H.: The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part IV. Lecture Notes in Computer Science, vol. 14923, pp. 139–173. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68385-5_5
12. Bariant, A., Bouvier, C., Leurent, G., Perrin, L.: Algebraic attacks against some arithmetization-oriented primitives. IACR Transactions on Symmetric Cryptology **2022**(3), 73–101 (2022). https://doi.org/10.46586/tosc.v2022.i3.73-101
13. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-hash. IACR Transactions on Symmetric Cryptology **2020**(S1), 88–131 (2020). https://doi.org/10.13154/tosc.v2020.iS1.88-131
14. Bellare, M., Rogaway, P.: Code-Based Game-Playing Proofs and the Security of Triple Encryption. Cryptology ePrint Archive, Paper 2004/331 (2004), https://eprint.iacr.org/2004/331, https://eprint.iacr.org/2004/331
15. Bellini, E., De Piccoli, A., Formenti, M., Gérault, D., Huynh, P., Pelizzola, S., Polese, S., Visconti, A.: Differential cryptanalysis with SAT, SMT, MILP, and CP: A detailed comparison for bit-oriented primitives. In: Deng, J., Kolesnikov, V., Schwarzmann, A.A. (eds.) CANS 23: 22nd International Conference on Cryptology and Network Security. Lecture Notes in Computer Science, vol. 14342, pp. 268–292. Springer, Singapore, Singapore, Augusta, GA, USA (Oct 31 – Nov 2, 2023). https://doi.org/10.1007/978-981-99-7563-1_13
16. Berger, T.P., Minier, M., Thomas, G.: Extended generalized Feistel networks using matrix representation. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8282, pp. 289–305. Springer Berlin Heidelberg, Germany, Burnaby, BC, Canada (Aug 14–16, 2014). https://doi.org/10.1007/978-3-662-43414-7_15
17. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 368–397. Springer Berlin Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_15
18. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS$^+$ signature framework. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 2129–2146. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3363229

19. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R., Viguier, B.: TurboSHAKE. Cryptology ePrint Archive, Report 2023/342 (2023), https://eprint.iacr.org/2023/342
20. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge Functions. Ecrypt Hash Workshop 2007 (May 2007)
21. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the Sponge construction. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer Berlin Heidelberg, Germany, Istanbul, Turkey (Apr 13–17, 2008). https://doi.org/10.1007/978-3-540-78967-3_11
22. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 313–314. Springer Berlin Heidelberg, Germany, Athens, Greece (May 26–30, 2013). https://doi.org/10.1007/978-3-642-38348-9_19
23. Beyne, T., Canteaut, A., Dinur, I., Eichlseder, M., Leander, G., Leurent, G., Naya-Plasencia, M., Perrin, L., Sasaki, Y., Todo, Y., Wiemer, F.: Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 299–328. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_11
24. Beyne, T., Verbauwhede, M.: Integral cryptanalysis in characteristic p. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025, Part I. Lecture Notes in Computer Science, vol. 16245, pp. 66–96. Springer, Singapore, Singapore, Melbourne, VIC, Australia (Dec 8–12, 2025). https://doi.org/10.1007/978-981-95-5018-0_3
25. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT'99. Lecture Notes in Computer Science, vol. 1592, pp. 12–23. Springer Berlin Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999). https://doi.org/10.1007/3-540-48910-X_2
26. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. Journal of Cryptology **18**(4), 291–311 (Sep 2005). https://doi.org/10.1007/s00145-005-0129-3
27. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) Advances in Cryptology – CRYPTO'90. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1991). https://doi.org/10.1007/3-540-38424-3_1
28. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Designs, Codes and Cryptography **70**(3), 369–383 (2014). https://doi.org/10.1007/s10623-012-9697-z
29. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy. pp. 353–367. IEEE Computer Society Press, London, United Kingdom (Apr 24–26, 2018). https://doi.org/10.1109/EuroSP.2018.00032
30. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of Keccak and Luffa. In: Joux, A. (ed.) Fast Software Encryption – FSE 2011.

Lecture Notes in Computer Science, vol. 6733, pp. 252–269. Springer Berlin Heidelberg, Germany, Lyngby, Denmark (Feb 13–16, 2011). https://doi.org/10.1007/978-3-642-21702-9_15
31. Bouvier, C., Briaud, P., Chaidos, P., Perrin, L., Salen, R., Velichkov, V., Willems, D.: New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and Jive compression mode. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 507–539. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38548-3_17
32. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 553–588 (2022). https://doi.org/10.46586/tches.v2022.i4.553-588
33. Buchmann, J.A., Dahmen, E., Hülsing, A.: XMSS - A practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 117–129. Springer Berlin Heidelberg, Germany, Tapei, Taiwan (Nov 29 – Dec 2 2011). https://doi.org/10.1007/978-3-642-25405-5_8
34. Campa, L., Roy, A.: Gröbner basis cryptanalysis of Anemoi. In: Fehr, S., Fouque, P.A. (eds.) Advances in Cryptology – EUROCRYPT 2025, Part I. Lecture Notes in Computer Science, vol. 15601, pp. 303–332. Springer, Cham, Switzerland, Madrid, Spain (May 4–8, 2025). https://doi.org/10.1007/978-3-031-91107-1_11
35. Cantor, D.G., Zassenhaus, H.: A new algorithm for factoring polynomials over finite fields. Mathematics of Computation **36**(154), 587–592 (1981). https://doi.org/10.1090/S0025-5718-1981-0606517-5
36. Cassiers, G.: MATCHI: formal verification of hardware private circuits. Cryptology ePrint Archive, Report 2025/1882 (2025), https://eprint.iacr.org/2025/1882
37. Cassiers, G., Masure, L., Momin, C., Moos, T., Standaert, F.X.: Prime-field masking in hardware and its soundness against low-noise SCA attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 482–518 (2023). https://doi.org/10.46586/tches.v2023.i2.482-518
38. Chen, K.Y., Chen, J.P.: Masking floating-point number multiplication and addition of Falcon first- and higher-order implementations and evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 276–303 (2024). https://doi.org/10.46586/tches.v2024.i2.276-303
39. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 430–448. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005). https://doi.org/10.1007/11535218_26
40. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Undergraduate Texts in Mathematics, Springer International Publishing, 4 edn. (2015). https://doi.org/10.1007/978-3-319-16721-3
41. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) Fast Software Encryption – FSE'97. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer Berlin Heidelberg, Germany, Haifa, Israel (Jan 20–22, 1997). https://doi.org/10.1007/BFb0052343

42. Dobraunig, C., Eichlseder, M., Groß, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked AES with fault countermeasures. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 315–342. Springer, Cham, Switzerland, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03329-3_11
43. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: Exploiting ineffective fault inductions on symmetric cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 547–572 (2018). https://doi.org/10.13154/tches.v2018.i3.547-572, https://tches.iacr.org/index.php/TCHES/article/view/7286
44. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Winning Submission to NIST Lightweight Cryptography (2021)
45. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**(3), 33 (Jul 2021). https://doi.org/10.1007/s00145-021-09398-9
46. Dods, C., Smart, N.P., Stam, M.: Hash based digital signature schemes. In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 3796, pp. 96–115. Springer Berlin Heidelberg, Germany, Cirencester, UK (Dec 19–21, 2005). https://doi.org/10.1007/11586821_8
47. Drake, J., Khovratovich, D., Kudinov, M.A., Wagner, B.: Hash-based multi-signatures for post-quantum Ethereum. IACR Communications in Cryptology (CiC) **2**(1), 13 (2025). https://doi.org/10.62056/aey7qjp10
48. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
49. Dziembowski, S., Faust, S., Skórski, M.: Optimal amplification of noisy leakages. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A: 13th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 9563, pp. 291–318. Springer Berlin Heidelberg, Germany, Tel Aviv, Israel (Jan 10–13, 2016). https://doi.org/10.1007/978-3-662-49099-0_11
50. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra **139**(1), 61–88 (1999). https://doi.org/10.1016/S0022-4049(99)00005-5
51. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. p. 75–83. ISSAC '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/780506.780516
52. Faugère, J.C., Gaudry, P., Huot, L., Renault, G.: Sub-cubic change of ordering for Gröbner basis: a probabilistic approach. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. p. 170–177. ISSAC '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2608628.2608669
53. Faugère, J.C., Gianni, P., Lazard, D., Mora, T.: Efficient computation of zero-dimensional Gröbner bases by change of ordering. Journal of Symbolic Computation **16**(4), 329–344 (1993). https://doi.org/10.1006/jsco.1993.1051

54. Faugère, J.C., Mou, C.: Sparse FGLM algorithms. Journal of Symbolic Computation **80**, 538–569 (2017). https://doi.org/10.1016/j.jsc.2016.07.025
55. Faust, S., Masure, L., Michel, E., Nguyen, H.H., Orlt, M., Standaert, F.X.: IP masking with generic security guarantees under minimum assumptions, and applications. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025, Part II. Lecture Notes in Computer Science, vol. 16246, pp. 544–575. Springer, Singapore, Singapore, Melbourne, VIC, Australia (Dec 8–12, 2025). https://doi.org/10.1007/978-981-95-5096-8_17
56. Faust, S., Masure, L., Micheli, E., Orlt, M., Standaert, F.X.: Connecting leakage-resilient secret sharing to practice: Scaling trends and physical dependencies of prime field masking. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024, Part IV. Lecture Notes in Computer Science, vol. 14654, pp. 316–344. Springer, Cham, Switzerland, Zurich, Switzerland (May 26–30, 2024). https://doi.org/10.1007/978-3-031-58737-5_12
57. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
58. Genêt, A.: On protecting SPHINCS+ against fault attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 80–114 (2023). https://doi.org/10.46586/tches.v2023.i2.80-114
59. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst meets fluid-SPN: Griffin for zero-knowledge applications. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 573–606. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38548-3_19
60. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations. IACR Transactions on Symmetric Cryptology **2024**(3), 44–83 (2024). https://doi.org/10.46586/tosc.v2024.i3.44-83
61. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021: 30th USENIX Security Symposium. pp. 519–535. USENIX Association (Aug 11–13, 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/grassi
62. Grassi, L., Khovratovich, D., Schofnegger, M.: Poseidon2: A faster version of the Poseidon hash function. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) AFRICACRYPT 23: 14th International Conference on Cryptology in Africa. Lecture Notes in Computer Science, vol. 14064, pp. 177–203. Springer, Cham, Switzerland, Sousse, Tunisia (Jul 19–21, 2023). https://doi.org/10.1007/978-3-031-37679-5_8
63. Grassi, L., Masure, L., Méaux, P., Moos, T., Standaert, F.X.: Generalized Feistel ciphers for efficient prime field masking. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024, Part III. Lecture Notes in Computer Science, vol. 14653, pp. 188–220. Springer, Cham, Switzerland, Zurich, Switzerland (May 26–30, 2024). https://doi.org/10.1007/978-3-031-58734-4_7
64. Guo, J., Liu, G., Song, L., Tu, Y.: Exploring SAT for cryptanalysis: (quantum) collision attacks against 6-round SHA-3. In: Agrawal, S., Lin, D. (eds.) Advances

in Cryptology – ASIACRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 645–674. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_22
65. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2011. Lecture Notes in Computer Science, vol. 6917, pp. 326–341. Springer Berlin Heidelberg, Germany, Nara, Japan (Sep 28 – Oct 1, 2011). https://doi.org/10.1007/978-3-642-23951-9_22
66. Hirose, S.: Sequential Hashing with Minimum Padding. Cryptogr. **2**(2), 11 (2018). https://doi.org/10.3390/CRYPTOGRAPHY2020011
67. Huelsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. RFC 8391 (May 2018). https://doi.org/10.17487/RFC8391, https://www.rfc-editor.org/info/rfc8391
68. Hülsing, A.: W-OTS+ - shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 13: 6th International Conference on Cryptology in Africa. Lecture Notes in Computer Science, vol. 7918, pp. 173–188. Springer Berlin Heidelberg, Germany, Cairo, Egypt (Jun 22–24, 2013). https://doi.org/10.1007/978-3-642-38553-7_10
69. Hülsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS$^+$. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
70. Hülsing, A., Kudinov, M.A.: Recovering the tight security proof of SPHINCS$^+$. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 3–33. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22972-5_1
71. Jahandideh, V., Mennink, B., Batina, L.: A decomposition approach for evaluating security of masking. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025, Part I. Lecture Notes in Computer Science, vol. 16245, pp. 580–612. Springer, Singapore, Singapore, Melbourne, VIC, Australia (Dec 8–12, 2025). https://doi.org/10.1007/978-981-95-5018-0_19
72. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) Fast Software Encryption – FSE'97. Lecture Notes in Computer Science, vol. 1267, pp. 28–40. Springer Berlin Heidelberg, Germany, Haifa, Israel (Jan 20–22, 1997). https://doi.org/10.1007/BFb0052332
73. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part II. Lecture Notes in Computer Science, vol. 8874, pp. 274–288. Springer Berlin Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014). https://doi.org/10.1007/978-3-662-45608-8_15
74. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press, New York, 3 edn. (2014). https://doi.org/10.1201/9781351133036
75. Khovratovich, D., Beltrán, M.M., Mennink, B.: Generic security of the SAFE API and its applications. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023, Part VIII. Lecture Notes in Computer Science, vol. 14445, pp. 301–327. Springer, Singapore, Singapore, Guangzhou, China (Dec 4–8, 2023). https://doi.org/10.1007/978-981-99-8742-9_10

76. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption – FSE'94. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer Berlin Heidelberg, Germany, Leuven, Belgium (Dec 14–16, 1995). https://doi.org/10.1007/3-540-60590-8_16
77. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 - Efficient short-input hashing for post-quantum applications. IACR Transactions on Symmetric Cryptology **2016**(2), 1–29 (2016). https://doi.org/10.13154/tosc.v2016.i2.1-29, https://tosc.iacr.org/index.php/ToSC/article/view/563
78. Kreuzer, M., Robbiano, L.: Computational Linear and Commutative Algebra. Springer International Publishing, Cham, 1 edn. (2016). https://doi.org/10.1007/978-3-319-43601-2
79. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography: Two Sides of One Tapestry. pp. 227–233. Springer US, Boston, MA (1994). https://doi.org/10.1007/978-1-4615-2694-0_23
80. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full Whirlpool compression function. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 126–143. Springer Berlin Heidelberg, Germany, Tokyo, Japan (Dec 6–10, 2009). https://doi.org/10.1007/978-3-642-10366-7_8
81. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (Oct 1979)
82. Lefevre, C., Beltrán, M.M., Mennink, B.: To Pad or Not to Pad? Padding-Free Arithmetization-Oriented Sponges. IACR Trans. Symmetric Cryptol. **2025**(1), 97–137 (2025). https://doi.org/10.46586/TOSC.V2025.I1.97-137
83. Lefevre, C., Mennink, B.: Tight preimage resistance of the Sponge construction. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 185–204. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15985-5_7
84. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
85. Masure, L., Méaux, P., Moos, T., Standaert, F.X.: Effective and efficient masking with low noise using small-Mersenne-prime ciphers. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 596–627. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30634-1_20
86. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) Advances in Cryptology – EUROCRYPT'93. Lecture Notes in Computer Science, vol. 765, pp. 386–397. Springer Berlin Heidelberg, Germany, Lofthus, Norway (May 23–27, 1994). https://doi.org/10.1007/3-540-48285-7_33
87. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. Cryptology ePrint Archive, Report 2003/161 (2003), https://eprint.iacr.org/2003/161
88. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali Hash-Based Signatures. RFC 8554 (Apr 2019). https://doi.org/10.17487/RFC8554, https://www.rfc-editor.org/info/rfc8554

89. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) Fast Software Encryption – FSE 2009. Lecture Notes in Computer Science, vol. 5665, pp. 260–276. Springer Berlin Heidelberg, Germany, Leuven, Belgium (Feb 22–25, 2009). https://doi.org/10.1007/978-3-642-03317-9_16
90. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer, New York, USA, Santa Barbara, CA, USA (Aug 20–24, 1990). https://doi.org/10.1007/0-387-34805-0_21
91. Moenck, R.T.: Fast computation of GCDs. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing. p. 142–151. STOC '73, Association for Computing Machinery, New York, NY, USA (1973). https://doi.org/10.1145/800125.804045
92. Moos, T., Saha, S., Standaert, F.X.: Prime masking vs. faults - exponential security amplification against selected classes of attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(4), 690–736 (2024). https://doi.org/10.46586/tches.v2024.i4.690-736
93. Müller, N., Moradi, A.: PROLEAD A probing-based hardware leakage detection tool. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 311–348 (2022). https://doi.org/10.46586/tches.v2022.i4.311-348
94. Nachef, V., Marrière, N., Volte, E.: Improved attacks on extended generalized Feistel networks. In: Foresti, S., Persiano, G. (eds.) CANS 16: 15th International Conference on Cryptology and Network Security. Lecture Notes in Computer Science, vol. 10052, pp. 562–572. Springer, Cham, Switzerland, Milan, Italy (Nov 14–16, 2016). https://doi.org/10.1007/978-3-319-48965-0_35
95. Naito, Y., Ohta, K.: Improved indifferentiable security analysis of PHOTON. In: Abdalla, M., De Prisco, R. (eds.) SCN 14: 9th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 8642, pp. 340–357. Springer, Cham, Switzerland, Amalfi, Italy (Sep 3–5, 2014). https://doi.org/10.1007/978-3-319-10879-7_20
96. NIST: Secure Hash Standard (SHS). National Institute of Standards and Technology, NIST FIPS PUB 180-4, U.S. Department of Commerce (Aug 2015). https://doi.org/10.6028/NIST.FIPS.180-4
97. NIST: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology, NIST FIPS PUB 202, U.S. Department of Commerce (Aug 2015). https://doi.org/10.6028/NIST.FIPS.202
98. NIST: Recommendation for Stateful Hash-Based Signature Schemes. National Institute of Standards and Technology, NIST SP 800-208 (Oct 2020). https://doi.org/10.6028/NIST.SP.800-208
99. NIST: Module-Lattice-Based Digital Signature Standard. National Institute of Standards and Technology, NIST FIPS PUB 204, U.S. Department of Commerce (Aug 2025). https://doi.org/10.6028/NIST.FIPS.204
100. NIST: Module-Lattice-Based Key-Encapsulation Mechanism Standard. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce (Aug 2025). https://doi.org/10.6028/NIST.FIPS.203
101. NIST: Stateless Hash-Based Digital Signature Standard. National Institute of Standards and Technology, NIST FIPS PUB 205, U.S. Department of Commerce (Aug 2025). https://doi.org/10.6028/NIST.FIPS.205

102. Nyberg, K.: Generalized Feistel networks. In: Kim, K., Matsumoto, T. (eds.) Advances in Cryptology – ASIACRYPT'96. Lecture Notes in Computer Science, vol. 1163, pp. 91–104. Springer Berlin Heidelberg, Germany, Kyongju, Korea (Nov 3–7, 1996). https://doi.org/10.1007/BFb0034838
103. Polygon: Plonky 3. GitHub (2025), https://github.com/Plonky3/Plonky3
104. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
105. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: Roy, B.K., Meier, W. (eds.) Fast Software Encryption – FSE 2004. Lecture Notes in Computer Science, vol. 3017, pp. 371–388. Springer Berlin Heidelberg, Germany, New Delhi, India (Feb 5–7, 2004). https://doi.org/10.1007/978-3-540-25937-4_24
106. Saarinen, M.J.O.: Accelerating SLH-DSA by two orders of magnitude with a single hash unit. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part I. Lecture Notes in Computer Science, vol. 14920, pp. 276–304. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68376-3_9
107. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
108. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press, Santa Fe, NM, USA (Nov 20–22, 1994). https://doi.org/10.1109/SFCS.1994.365700
109. Steiner, M.J.: Gröbner basis cryptanalysis of Ciminion and Hydra. IACR Transactions on Symmetric Cryptology **2025**(1), 240–275 (2025). https://doi.org/10.46586/tosc.v2025.i1.240-275
110. Storjohann, A.: Algorithms for matrix canonical forms. Ph.D. thesis, ETH Zurich (2000). https://doi.org/10.3929/ethz-a-004141007
111. Suzaki, T., Minematsu, K.: Improving the generalized Feistel. In: Hong, S., Iwata, T. (eds.) Fast Software Encryption – FSE 2010. Lecture Notes in Computer Science, vol. 6147, pp. 19–39. Springer Berlin Heidelberg, Germany, Seoul, Korea (Feb 7–10, 2010). https://doi.org/10.1007/978-3-642-13858-4_2
112. Szepieniec, A., Lemmens, A., Sauer, J.F., Threadbare, B.: The Tip5 hash function for recursive STARKs. Cryptology ePrint Archive, Report 2023/107 (2023), https://eprint.iacr.org/2023/107
113. The OSCAR Team: OSCAR – Open Source Computer Algebra Research system, Version 1.4.1 (2025), https://www.oscar-system.org
114. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.6) (2025), https://www.sagemath.org
115. Williams, V.V., Xu, Y., Xu, Z., Zhou, R.: New bounds for matrix multiplication: from alpha to omega. In: Woodruff, D.P. (ed.) 35th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 3792–3835. ACM-SIAM, Alexandria, VA, USA (Jan 7–10, 2024). https://doi.org/10.1137/1.9781611977912.134

116. Yang, H.S., Zheng, Q.X., Yang, J., feng Liu, Q., Tang, D.: A new security evaluation method based on resultant for arithmetic-oriented algorithms. In: Chung, K.M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, Part VII. Lecture Notes in Computer Science, vol. 15490, pp. 457–489. Springer, Singapore, Singapore, Kolkata, India (Dec 9–13, 2024). https://doi.org/10.1007/978-981-96-0941-3_15

Appendix

A Algorithms for Sponge

Algorithm 1 Tweakable sponge function TS.

Require: $(\mathtt{IV}, m, t) \in \mathbb{F}_p^n \times \mathbb{F}_p^* \times \mathbb{F}_p^\tau$ and $\mathsf{TP} : \mathbb{F}_p^n \times \mathbb{F}_p^\tau \to \mathbb{F}_p^n$ tweakable permutation
Ensure: $h \in \mathbb{F}_p^d$.

$S \leftarrow \mathtt{IV}$
$h \leftarrow \emptyset$
$\{(m_1, \ldots, m_\ell), C\} \leftarrow \mathsf{padBlock}_r(m)$
for $1 \leq i \leq \ell - 1$ **do**
    $S \leftarrow \mathsf{TP}\,(S + (m_i \parallel 0^*, t))$
end for
$S \leftarrow \mathsf{TP}\,(S + (m_\ell \parallel 0^*) + C, t)$
for $1 \leq i \leq \lceil d/r \rceil$ **do**
    $h \leftarrow h \parallel \lfloor S \rfloor_r$
    $S \leftarrow \mathsf{TP}(S, t)$
end for
return $h[1 : d]$

Algorithm 2 Definition of $\mathsf{padBlock}$ (usually, $\{\rho_0, \rho_1\} = \{0, 1\}$)

Function $\mathsf{padBlock}_r$
Require: $m \in \mathbb{F}_p^*$, $\{\rho_0, \rho_1\} \in \mathbb{F}_p^c$ with $\rho_0 \neq \rho_1$
Ensure: $(\overline{m}, C) \in (\mathbb{F}_p^r)^* \times \{\rho_0, \rho_1\}$

if $|m| \mod r \neq 0$ **then**
    $m \leftarrow m \parallel 1 \parallel 0^{(|m| \mod r)-1}$
    $C \leftarrow \rho_0$
else
    $C \leftarrow \rho_1$
end if
for $0 \leq i < |m|/r$ **do**
    $m_i \leftarrow m[r \cdot i : r \cdot (i + 1) - 1]$
end for
return $\{(m_0, \ldots, m_{|m|/r-1}), C\}$

B Round Constants Generation

Round constants are generated in a similar manner as for mid-pSquare. We denote the bit-wise rotation left via $\lll$. The first 384 bits of the binary sequence of π,

split into 6 chunks of 64 bits, are given by

$$\pi_{\text{bin64},0} = \texttt{0xC90FDAA22168C234}; \qquad \pi_{\text{bin64},1} = \texttt{0xC4C6628B80DC1CD1};$$
$$\pi_{\text{bin64},2} = \texttt{0x29024E088A67CC74}; \qquad \pi_{\text{bin64},3} = \texttt{0x020BBEA63B139B22};$$
$$\pi_{\text{bin64},4} = \texttt{0x514A08798E3404DD}; \qquad \pi_{\text{bin64},5} = \texttt{0xEF9519B3CD3A431B}.$$

The round constants for round i, namely $c_{i,j}, c'_{i,j} \in \mathbb{F}_{2^{31}-1},\ \forall\ 0 \leq j \leq (\frac{n}{4} - 1)$, are then defined as:

- $c_{i,j} = (\pi_{\text{bin64},j} \lll i) \bmod 2^{31}$;
- $c'_{i,j} = \big(\pi_{\text{bin64},j} \lll (i + 32)\big) \bmod 2^{31}$.

As no sequence of 31 consecutive 1s exists in $\pi_{\text{bin64},j}$, it holds that $c_{i,j}, c'_{i,j} \in \mathbb{F}_{2^{31}-1}$. Overall, this round constant generation enables efficient on-the-fly computation of all constants from 6 words of 64-bit and follows the nothing-up-my-sleeve paradigm by being based on a natural constant.

C Tool for Automated Differential Search

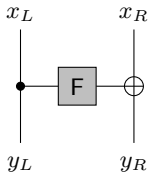
Automated tools have a long history in differential cryptanalysis. Due to the Boolean nature of classical ciphers, differential trail searches can be reformulated as Boolean satisfiability or optimization problem, and state-of-the-art solvers like SAT, SMT, MILP, etc. are used to find efficient trails. For a comparative analysis of various solvers in differential cryptanalysis we refer the reader to [15].

In theory, any differential trail search for pSquare-hash could be translated to a SAT/MILP problem. However, we do not expect this approach to be feasible, since translating algebraic relations from $\mathbb{F}_p$ to Boolean relations will incur a non-negligible overhead to the solver. Also, we recall that our non-linear layer operates over $\mathbb{F}_p^2$, so it operate on $\lceil 2 \cdot \log_2(p) \rceil = 62$ bits, which is also expected to negatively impact solver performance. For example, finding an input-output pair whose difference activates exactly one square, see Figure 4, has an expected (enumeration) cost of $\mathcal{O}(p)$. Henceforth, we consider the extension of SAT, SMT, MILP, etc. tools to our setting as infeasible.

On the other hand, counting the number of active squares in a differential trail for pSquare-hash is a relatively simple task, because we can do it *symbolically*. To explain this process let us consider the simple Feistel in Figure 6a, which represents one non-linear layer of pSquare-hash over $\mathbb{F}_p^4$. Suppose we have the input difference $(\Delta x_L, \Delta x_R) = (\delta, 0)$, where $\delta \in \mathbb{F}_p^2 \backslash \{0\}$, then possible transitions $\delta \xrightarrow{\mathsf{F}} \hat{\delta} \neq 0$ through F activate either 1 or 2 squares, see Figure 4. Also recall that F is a permutation of $\mathbb{F}_p^2$, so transitions $\delta \xrightarrow{\mathsf{F}} 0$ are impossible. Likewise, if we have the input difference $(\Delta x_L, \Delta x_R) = (\delta_L, \delta_R)$, where $\delta_L, \delta_R \in \mathbb{F}_p^2 \setminus \{0\}$, then we have the transitions

$$\delta_L \xrightarrow{\mathsf{F}} \hat{\delta}_L + \delta_R \neq 0, \qquad \delta \xrightarrow{\mathsf{F}} \hat{\delta}_L + \delta_R = 0. \tag{17}$$

However, this time both cases are achievable with 1 active square in F only.



(a) Non-linear layer of pSquare-hash over $\mathbb{F}_p^4$. A wire represents an element in $\mathbb{F}_p^2$.

Δx_L	Δx_R	Δy_L	Δy_R	Active Squares
0	0/1/∗	0	0/1/∗	0
∗	0/1	∗	∗	0
1	0	1	1	1, 2
1	∗	1	∗	0
1	1	1	1	1, 2
1	1	1	0	1, 2

(b) Propagation rules.

Figure 6. Propagation rules through one non-linear layer of pSquare-hash. "0" represents a known zero difference, "1" represents a known non-zero difference, "∗" a truncated difference.

The computational benefit of our symbolic approach comes from the fact that we do not have to introduce new variables after every application of F, for active square enumeration it is sufficient to track whether a state has

- a known zero difference,
- a known difference, or
- an unknown, i.e. truncated, difference.

In Table 6b we list all propagation rules for F together with the number of active squares.

Likewise, we derive the propagation rules for the linear layer of pSquare-hash:

- if two known difference are added we yield a known difference;
- if two known non-zero differences are added we yield a known difference or a zero difference;
- if a known difference is added to a truncated difference we yield a truncated difference.

Translating all these considerations into an automated truncated differential search tool for pSquare-hash is straight-forward. Starting from an input difference state $\Delta \in \{0, 1, *\}^m$, in the first round we simply enumerate all possible propagations through the non-linear and linear layer respectively. Next we repeat this procedure for every non-fully truncated differential trail in our enumeration.

A downside of this approach is that we do not obtain actual differential trails, only *hypothetical* ones. So, for every found trail by our tool we have to check manually if it is possible to translate it into a possible (symbolic) trail. Moreover, our approach is not exhaustive over the space of all truncated differentials. In principle, linear combinations of truncated differences could cancel with a certain probability, but our tool is not capable of covering such cases. An example of such a truncated trail is given in Equation (16).

Table 11. Truncated differentials for $n = 16$ (1/2).

(a) Probability 1.

Differentials	
$(0,0,0,0 \mid 0,\delta,0,0) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta,0,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,*,\delta \mid 0,\delta,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,C\delta,* \mid 0,0,*,\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid *,*,C\delta,*)$	$(0,0,0,0 \mid 0,0,\delta,0) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,\delta,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,0,\delta \mid 0,0,\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,C\delta,*,* \mid 0,*,0,\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid *,C\delta,*,*)$
$(0,0,0,0 \mid 0,\delta,-\delta,0) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta,-\delta,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,0 \mid 0,\delta,-\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid 0,*,*,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$	$(0,0,0,0 \mid 0,\delta,0,-C^{-1}\delta) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta,0,-C^{-1}\delta \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,-\delta,*,0 \mid 0,\delta,0,-C\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,-C^{-1}\delta,*,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$
$(0,0,0,0 \mid 0,0,\delta,-C^{-1}\delta) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,\delta,-C^{-1}\delta \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,-\delta,0 \mid 0,0,\delta,-C^{-1}\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,*,-\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$	Condition: $\delta_1 + \delta_2 + C\delta_3 = 0$, $(0,0,0,0 \mid 0,\delta_1,\delta_2,\delta_3) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta_1,\delta_2,\delta_3 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,0 \mid 0,\delta_1,\delta_2,\delta_3)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,*,*,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$,

(b) Probability $1/p$.

Differentials	
Condition: $\delta_1 = \begin{pmatrix}0\\y\end{pmatrix}$, $\delta_2 = \begin{pmatrix}-y\\-y\end{pmatrix}$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} \delta_2$, $C\delta_1 + \delta_2 = 0$, $(0,0,0,0 \mid 0,\delta_1,0,0) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta_1,0,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,\delta_2,\delta_1 \mid 0,\delta_1,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,C\delta_1,0 \mid 0,0,\delta_2,\delta_1)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,*,C\delta_1,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$	Condition: $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} \delta_2$, $(0,0,0,0 \mid 0,0,\delta_1,0) \xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,\delta_1,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,\delta_2,0,\delta_1 \mid 0,0,\delta_1,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,C\delta_1,*,0,0 \mid \delta_2,0,\delta_1)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,C\delta_1,*,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$
Condition: $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} \delta_2$, $(0,\delta_1,0,0 \mid 0,0,-\delta_2,-\delta_1) \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,0,0 \mid 0,\delta_1,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta_1,0,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,*,\delta_1 \mid 0,\delta_1,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,C\delta_1,* \mid 0,0,*,\delta_1)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid *,*,C\delta_1,*)$	Condition: $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} \delta_2$, $(0,0,\delta_1,0 \mid 0,-\delta_2,0,-\delta_1) \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,0,0 \mid 0,0,\delta_1,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,\delta_1,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,0,\delta_1 \mid 0,0,\delta_1,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,C\delta_1,*,* \mid 0,*,0,\delta_1)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid *,C\delta_1,*,*)$

(c) Probability $1/p^2$.

Differentials	
Condition: $\delta = \begin{pmatrix}0\\y\end{pmatrix}$, $\delta \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} -C\delta = \begin{pmatrix}-y\\-y\end{pmatrix}$, $(0,\delta,0,0 \mid 0,0,C\delta,-\delta) \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,0,0 \mid 0,\delta,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,\delta,0,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,-C\delta,\delta \mid 0,\delta,0,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,C\delta,0 \mid 0,0,-C\delta,\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,*,C\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$	Condition: $\delta = \begin{pmatrix}0\\y\end{pmatrix}$, $\delta \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{F}} -C\delta = \begin{pmatrix}-y\\-y\end{pmatrix}$, $(0,0,\delta,0 \mid 0,C\delta,0,-\delta) \xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,0,0,0 \mid 0,0,\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,0,\delta,0 \mid 0,0,0,0)$ $\xrightarrow[\text{prob. } \frac{1}{p}]{\mathsf{R}} (0,-C\delta,0,\delta \mid 0,0,\delta,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,C\delta,*,0 \mid 0,-C\delta,0,\delta)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (0,*,*,* \mid *,C\delta,*,0)$ $\xrightarrow[\text{prob. } 1]{\mathsf{R}} (*,*,*,* \mid 0,*,*,*)$

Table 12. Truncated differentials for $n = 16$ (2/2).

(a) Probability $1/p^3$.

Differentials	
Condition: $\delta_1 = \binom{0}{\delta}$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} -\delta_2$, $-C\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} -\delta_3$, $-\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} C\delta_1$,	Condition: $\delta_1 = \binom{0}{\delta}$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} -\delta_2$, $-C\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} -\delta_3$, $-\delta_1 \xrightarrow[\text{prob. } \frac{1}{2}]{\text{F}} C\delta_1$,
$(0, -C\delta_1, 0, \delta_1 \mid -\delta_2, -C\delta_1, -C\delta_1 - \delta_2 - \delta_1, 0) \xrightarrow[\text{prob. } \frac{1}{2^{16}}]{\text{R}} (0, 0, -\delta_1, 0 \mid 0, -C\delta_1, 0, \delta_1)$	$(0, 0, -C\delta_1, \delta_1 \mid -\delta_2, -\delta_1 - C\delta_1 + \delta_2, -C\delta_1, 0) \xrightarrow[\text{prob. } \frac{1}{2^{16}}]{\text{R}} (0, -\delta_1, 0, 0 \mid 0, 0, -C\delta_1, \delta_1)$
$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, 0 \mid 0, 0, -\delta_1, 0)$	$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, 0 \mid 0, -\delta_1, 0, 0)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (0, 0, -\delta_1, 0 \mid 0, 0, 0, 0)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (0, -\delta_1, 0, 0 \mid 0, 0, 0, 0)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (0, *, 0, -\delta_1 \mid 0, 0, -\delta_1, 0)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (0, 0, *, -\delta_1 \mid 0, -\delta_1, 0, 0)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (*, -C\delta_1, *, * \mid 0, *, 0, -\delta_1)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (*, *, -C\delta_1, * \mid 0, 0, *, -\delta_1)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (*, *, *, * \mid *, -C\delta_1, *, *)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (*, *, *, * \mid *, *, -C\delta_1, *)$

(b) Probability $1/p^4$.

Differentials	
Conditions: $\delta_1 = \binom{\delta}{\delta}$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\text{F}} -\delta_2$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\text{F}} -\delta_1$, $\delta_1 \xrightarrow[\text{prob. } \frac{1}{p}]{\text{F}} \delta_3$, $C\delta_3 \xrightarrow[\text{prob. } \frac{1}{p}]{\text{F}} \delta_4$,	
$(0, 0, 0, \delta_1 \mid \delta_1 + \delta_2, -C\delta_1, -C\delta_1, -C\delta_1) \xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (\delta_1, 0, 0, 0 \mid 0, 0, 0, \delta_1)$	$(0, 0, 0, \delta_1 \mid \delta_1 + \delta_2, -C\delta_1, -C\delta_1, -C\delta_1) \xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (\delta_1, 0, 0, 0 \mid 0, 0, 0, \delta_1)$
$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, 0 \mid \delta_1, 0, 0, 0)$	$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, 0 \mid \delta_1, 0, 0, 0)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (\delta_1, 0, 0, 0 \mid 0, 0, 0, 0)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (\delta_1, 0, 0, 0 \mid 0, 0, 0, 0)$
$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, \delta_3 \mid \delta_1, 0, 0, 0)$	$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (0, 0, 0, \delta_3 \mid \delta_1, 0, 0, 0)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (*, C\delta_3, C\delta_3, C\delta_3 \mid 0, 0, 0, \delta_3)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (*, C\delta_3, C\delta_3, C\delta_3 \mid 0, 0, 0, \delta_3)$
$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (*, C^2\delta_3 + \delta_4, *, * \mid *, C\delta_3, C\delta_3, C\delta_3)$	$\xrightarrow[\text{prob. } \frac{1}{p}]{\text{R}} (*, *, C^2\delta_3 + \delta_4, * \mid *, C\delta_3, C\delta_3, C\delta_3)$
$\xrightarrow[\text{prob. } 1]{\text{R}} (*, *, *, * \mid *, C^2\delta_3 + \delta_4, *, *)$	$\xrightarrow[\text{prob. } 1]{\text{R}} (*, *, *, * \mid *, *, C^2\delta_3 + \delta_4, *)$

C.1 Truncated Differentials

In this section we provide examples of truncated differentials for pSquare-hash's round function for $n = 16$. Note that they can be extended to $n = 24$ in a natural fashion. For ease of presentation let $C = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$.

Starting with probability 1 truncated differentials, for $n = 16$ we found 6 trails (Table 11a) and for $n = 24$ we found 40 trails. For both cases, all found truncated differentials cover at most 4 rounds.

For probability $1/p$ truncated differentials, we found 4 trails for $n = 16$ (Table 11b) and 8 for $n = 24$. For both cases, all found truncated differentials cover at most 5 rounds.

For probability $1/p^2$ truncated differentials, we found 2 trails for $n = 16$ (Table 11c) and 4 for $n = 24$. For both cases, all found truncated differentials cover at most 6 rounds.

For probability $1/p^3$ truncated differentials, we found ≤ 66 trails for $n = 16$ and ≤ 142 for $n = 24$. For both cases, all found truncated differentials cover at most 6 rounds. In Table 12a we present two examples for $n = 16$.

For probability $1/p^4$ truncated differentials, we found ≤ 38 trails for $n = 16$ and ≤ 73 for $n = 24$. For both cases, all found truncated differentials cover at most 7 rounds. In Table 12b we present two examples for $n = 16$.

D Details about Statistical Cryptanalysis

D.1 About the Tweak Schedule

Here, we limit ourselves to point out some statistical properties of the tweak schedule.

We note that (1st) the only fixed point of T'_S is $p-1$, and that (2nd) $\Lambda(x) \oplus (\Lambda(x) \gg 1)$ over $\mathbb{F}_2^{31}$ corresponds to $3 \cdot x$ over $\mathbb{F}_p$ for x whose bit-representation is given by $(x_0, 0, x_2, 0, x_3, 0, \ldots, 0, x_{15}, 0) \in \{0,1\}^{31}$. At the same time, this simple algebraic representation is destroyed by the XOR-addition with $\Lambda\left(2^{30}-1\right)$ (which is not of the form just given).

The matrix $\mathrm{circ}(2, 1, \ldots, 1)$ provides full diffusion over 1 round, but admits several invariant subspaces such as $\langle [1, -1, 0, 0, 0, \ldots, 0] \rangle$, $\langle [1, 0, -1, 0, 0, \ldots, 0] \rangle$, $\ldots \subseteq \mathbb{F}_p^n$, and $\langle [1, 1, 1, \ldots, 1] \rangle \subseteq \mathbb{F}_p^n$. At the same time, it is very unlikely that these subspace trails do survive for more than two rounds of the tweak schedule, as (1st) T'_S is not a linear function over $\mathbb{F}_p$, and (2nd) due to the round constant addition. In particular, since $-x = \Lambda^{-1}\big(\Lambda(x) \oplus \mathbf{1}\big)$ over $\mathbb{F}_{2^{31}-1}$, note that

$$\lambda\big(\mathsf{T}'_S(x)\big) = x_0 \| x_0 \oplus x_1 \oplus 1 \| x_1 \oplus x_2 \oplus 1 \| \ldots \| x_{30} \oplus x_{31} \oplus 1\,, \tag{18}$$

$$\lambda\big(-\mathsf{T}'_S(-x)\big) = x_0 \| x_0 \oplus x_1 \| x_1 \oplus x_2 \| \ldots \| x_{30} \oplus x_{31}\,, \tag{19}$$

which implies that $\mathsf{T}'_S(x) \neq -\mathsf{T}'_S(-x)$ for each $x \in \mathbb{F}_{2^{31}-1}$.

D.2 Other Statistical Attacks

In a linear attack [6, 86], 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. Impossible differentials attacks [25,26] exploit (truncated) differentials that hold with probability 0. They are usually set up by concatenating two consecutive truncated differentials that hold with probability 1 and that do not match in the middle. Since 4 rounds are sufficient to achieve full diffusion, and following the analysis provided in [16, §3.2] we claim that there exist no impossible differential that can cover more than $2 \cdot 4 + 2 = 10$ ($\tau = 0$) and $2 \cdot 8 + 2 \cdot \lceil \tau / n \rceil \in \{18, 20\}$ ($\tau \neq 0$) rounds. Similar conclusion holds for zero-linear correlation [28] and square/integral attacks [41].

In [94], Nachef et al. presented distinguishers for EGFN schemes based on the mean value and of the variance of the probabilistic distribution of the images of a set of input texts that satisfy particular linear relations. By adapting their attack to our scheme, and based on [94, Table 6], it is possible to distinguish between 7 rounds of the pSquare-hash permutation (with $n = 16$) and a random permutation using $p^4 \approx 2^{124}$ inputs (we limit ourselves to mention that it is possible to distinguish more rounds if more texts are used). A similar result holds for $n = 24$. At the current state of the art, we emphasize that no (second-)preimage and/or collision attack based on such distinguisher is known.

E Details – Growth of the Degree

Table 13. Maximum and minimum degree of Log_2 of pSquare-hash for $r \geq 14$ rounds.

(a) $n = 8$.

round	$\log_2$(min degree)	$\log_2$(max degree)
14	26.00000000000000	28.00000000000000
15	28.00000000000000	30.00000000000000
16	30.00000000000000	31.58496249982541
17	31.58496249982541	32.58496249982542
18	32.58496249982542	33.32192809381247
19	33.24792751231212	33.72792045324493
20	33.70043971690083	33.91886323594208
21	33.90689059435448	33.98013957627692
22	33.97727992213498	33.99506046568450
23	33.99435343550997	33.99876670001675
24	33.99859042840040	33.99969177282251
25	33.99964773518443	33.99992294837124
26	33.99991194085172	33.99998073647088
27	33.99997798470910	33.99999518313412
28	33.99999449520106	33.99999879477733
29	33.99999862279452	33.99999969768672
30	33.99999965469105	33.99999992341397
31	33.99999991266506	33.99999997984578
32	33.99999997715855	33.99999999395374
33	33.99999999328193	33.99999999748072
34	33.99999999731277	33.99999999832048
35	33.99999999832048	33.99999999848843

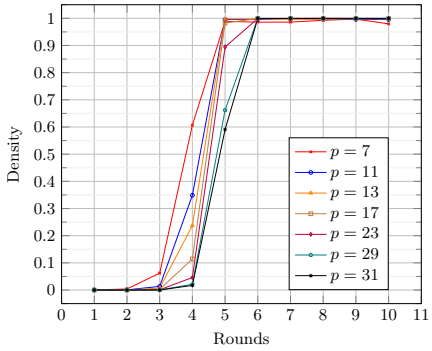
(b) $n = 12$.

round	$\log_2$(min degree)	$\log_2$(max degree)
14	26.00000000000000	28.00000000000000
15	28.00000000000000	30.00000000000000
16	30.00000000000000	31.58496249982541
17	31.58496249982541	32.58496250027329
18	32.58496250027329	33.58496249982542
19	33.58496249971345	34.16992500017334
20	34.12928301571652	34.44294349461322
21	34.44294349455144	34.55074678400738
22	34.55074678400738	34.57648434544532
23	34.57648434544532	34.58284762720468
24	34.58284762720468	34.58443407190973
25	34.58443407190973	34.58483041065492
26	34.58483041065492	34.58492947833065
27	34.58492947833065	34.58495424418668
28	34.58495424418668	34.58496043558425
29	34.58496043558425	34.58496198342949
30	34.58496198342949	34.58496237039055
31	34.58496237039055	34.58496246713079
32	34.58496246713079	34.58496249131586
33	34.58496249131586	34.58496249736212
34	34.58496249736212	34.58496249887369
35	34.58496249887369	34.58496249920959

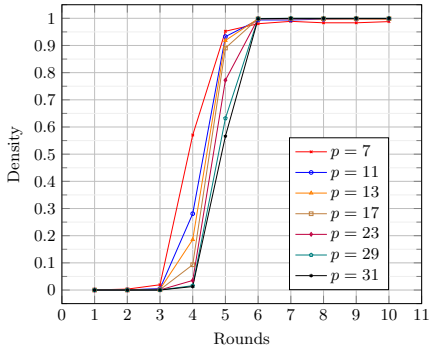
(c) $n = 16$.

round	$\log_2$(min degree)	$\log_2$(max degree)
14	26.0	28.0
15	28.0	30.0
16	30.0	31.584962499825412
17	31.584962499825412	32.584962500273285
18	32.584962500273285	33.80735492109788
19	33.584962499825416	34.45943161735476
20	34.32192809374529	34.794415865188064
21	34.741466985195665	34.951284713620645
22	34.924812502322986	34.98797452294129
23	34.98156728054212	34.997003013926395
24	34.995413850877256	34.99925133593967
25	34.99885482774912	34.99981286939651
26	34.999713791137474	34.99995321861688
27	34.99992845209951	34.99998830378872
28	34.999982112349805	34.999997074948354
29	34.99999552710053	34.99999926772993
30	34.99999888076872	34.99999981592481
31	34.99999971918455	34.999999952973496
32	34.999999928788434	34.99999998723566
33	34.9999999811894	34.999999995801204
34	34.99999999428964	34.99999999794259
35	34.9999999975647	34.99999999844645

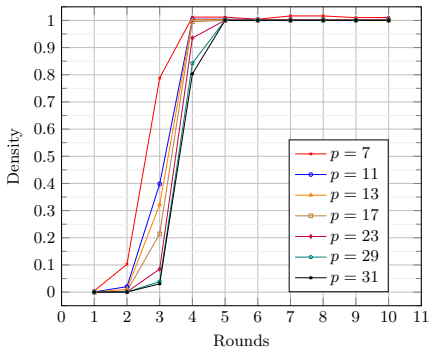
F Density



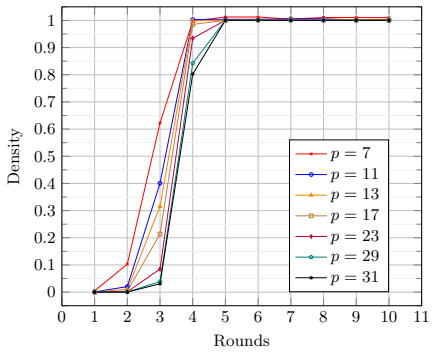
(a) Minimum average density with initial matrix.



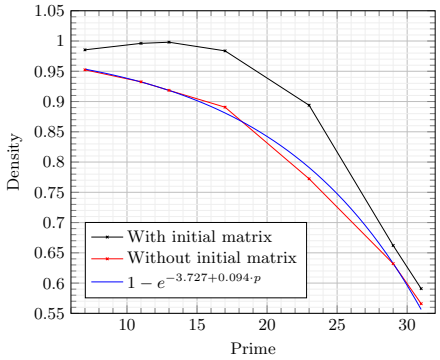
(b) Minimum average density without initial matrix.



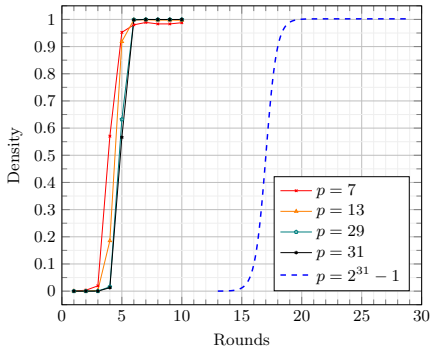
(c) Maximum average density with initial matrix.



(d) Maximum average density without initial matrix.

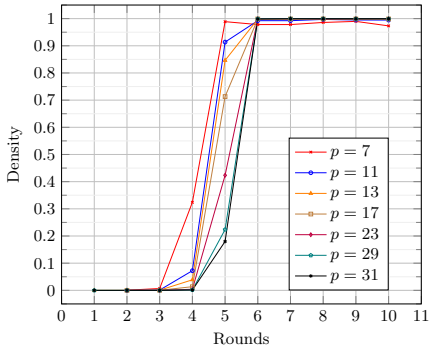


(e) Minimum average density after 5 rounds vs. prime.

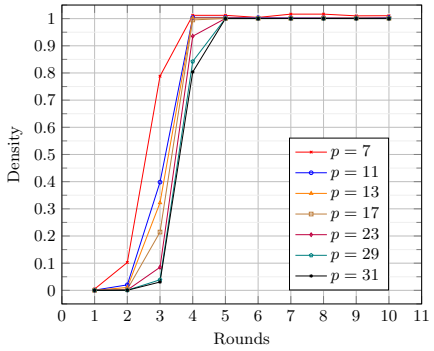


(f) Extrapolation of minimum average density without initial matrix for $p = 2^{31} - 1$.

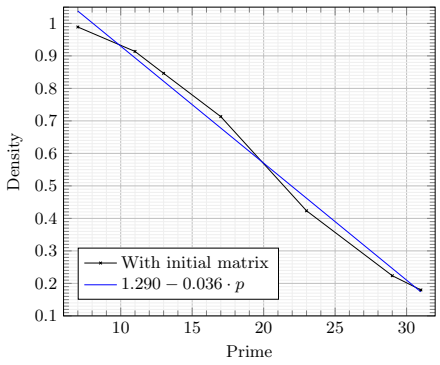
Figure 7. Minimum and maximum average density of the interpolation polynomial of toy pSquare-hash sponge with $n = 8$ and rate 4 for primes $7 \leq p \leq 31$. All Feistel constants have been set to zero.



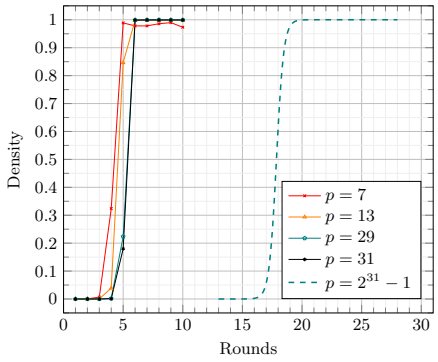
(a) Minimum average density with initial matrix.



(b) Maximum average density with initial matrix.



(c) Minimum average density after 5 rounds vs. prime.



(d) Extrapolation of minimum average density with initial matrix for $p = 2^{31} - 1$.

Figure 8. Minimum and maximum average density of the interpolation polynomial of toy pSquare-hash sponge with $n = 16$ and rate 4 for primes $7 \leq p \leq 31$. All Feistel constants have been set to zero.

G Details about Algebraic Cryptanalysis

In this section, we show that 12 blocks for $\tau = 0$ and 14 blocks otherwise (respectively, $4 \cdot 12 = 48$ and $4 \cdot 14 = 56$ rounds) are sufficient to prevent algebraic attacks such as the interpolation attack [72], and the higher-order differential attack [76, 79].

G.1 Interpolation Attack

The interpolation attack aims to re-construct the polynomial that defines the attacked scheme, and exploiting it to break the scheme. The simplest way to prevent the interpolation attack is to ensure that the polynomial representation is of maximum degree and dense. This implies that the attacker needs (more than) the full-codebook to construct the polynomial, making the attack infeasible. (We recall that a Meet-in-the-Middle version of this attack is feasible.)

In the case of pSquare-hash with $\tau = 0$, then 24 rounds (equivalently, 6 blocks) are sufficient to reach close to the maximum degree and density in one direction. Due to the MitM approach, we conjecture that $6 \cdot 2 = 12$ blocks (equivalently, 48 rounds) are sufficient to prevent algebraic attacks based on interpolation attacks. In the case of pSquare-hash with $\tau \neq 0$, based on the algebraic complexity of both the polynomial systems of the scheme and of the tweak schedule, we add 2 extra blocks (equivalently, 8 rounds) to mitigate related-tweak variants of the interpolation attack. We recall that 8 rounds guarantee a complex algebraic representation of the tweak schedule, which is represented by a polynomial of maximum degree and dense.

G.2 Higher-Order Differential Attack

In the case of a binary function, higher-order differential attacks exploit the fact that given a function F over $\mathbb{F}_2^n$ of degree δ, then $\sum_{x \in \mathcal{V}} \mathsf{F}(x) = 0$ for any affine subspace $\mathcal{V} \subseteq \mathbb{F}_2^n$ of dimension at least $\delta + 1$ (that is, $\dim(\mathcal{V}) \geq \deg(\mathsf{F}) + 1$). An analogous property holds for ciphers over $\mathbb{F}_p^n$, as shown in [23]. Given a function F over $\mathbb{F}_p^n$ of degree strictly less than $d \cdot (p-1)$, and given any affine subspace $\mathfrak{V} \subseteq \mathbb{F}_p^n$ of dimension d, then

$$\sum_{x \in \mathcal{V}} x = \sum_{x \in \mathcal{V}} \mathsf{F}(x) = 0\,.$$

Recently, Beyne and Verbauwhede [24] extended such a result by introducing "ultrametric integral cryptanalysis" over finite rings of prime characteristic p.

As we saw before, 34 rounds are necessary for pSquare-hash in order to reach maximum degree, which implies that the attack cannot be performed for the case of pSquare-hash instantiated with 48 rounds. Similar to how we handle the previous algebraic attack, we add 8 extra rounds for $\tau \neq 0$ to mitigate related-tweak variants of the higher-order differential attack.

G.3 (We Do Not Care About) Zero-Sum Partitions

Another property that can be demonstrated for some inner primitive in a hash function (with a relatively low degree) is based on the *zero-sum partition*. This direction has been investigated e.g. in [30] for two SHA-3 candidates, Luffa and KECCAK. More generally, a *zero-sum structure* for a function F is defined as a set $\mathcal{Z} = \{z_i\}_i$ that sum to zero, and for which the corresponding outputs $\mathsf{F}(z_i)$ also sum to zero, i.e., $\sum_i z_i = \sum_i \mathsf{F}(z_i) = 0$. For an iterated function, the existence of zero sums is usually due to the particular structure of the round function or to a low degree. Since it is expected that a randomly chosen function does not have many zero sums, the existence of several such sets can be seen as a distinguishing property of the internal function.

Definition 4 (Zero-Sum Partition [30]). *A zero-sum partition for* F *over* $\mathbb{F}_p^t$ *of size* m *is a collection of* m *disjoint sets* $\{\mathcal{Z}_0, \mathcal{Z}_1, \ldots, \mathcal{Z}_{m-1}\}$ *such that*

(1) $\bigcup_i \mathcal{Z}_i = \mathbb{F}_p^t$, *and*
(2) each $\mathcal{Z}_i$ *is a zero-sum set (that is,* $\sum_{x \in \mathcal{Z}_i} x = \sum_{x \in \mathcal{Z}_i} \mathsf{F}(x) = 0$*).*

Here we explicitly state that *we do* ***not*** *make claims about the security of* pSquare-hash *against zero-sum partitions*. This choice is motivated by fact that it is unclear how to convert a zero-sum partition of the permutation that instantiates the compression/hash function in a concrete (pre-image or collision) attack against the compression/hash function itself. Focusing on e.g. SHA-3/Keccak, this is even more evident when considering the the gap present in the literature between the number of rounds of the internal permutation that can be covered by a zero-sum partition and by the number of rounds in the corresponding sponge hash function that can be broken e.g. via a preimage or a collision attack, respectively, 24 [30] and 6 [64] rounds.

G.4 Linearization Attack

Let $\mathsf{F}_1, \ldots, \mathsf{F}_m \in \mathbb{F}_p[x_1, \ldots, x_n]$ be polynomials, one approach to solve $\mathsf{F}_1 = \ldots = \mathsf{F}_m = 0$ is via linearization of the system. Every monomial present the equations is substituted with a new variable until the polynomials are linear. Then, one performs e.g. Gaussian elimination to solve the system. The complexity of Gaussian elimination [110] scales with $\mathcal{O}\big((\#\text{Monomials})^\omega\big)$, where $2 \leq \omega < 2.371552$ [115] is a linear algebra constant.

While this attack is effective in the case of a cipher/keyed primitive, we recall that it is not a competitive attack strategy in the case of a hash function. Indeed, in the case of a cipher, the variables are related to the key only, which implies that one can collect as many plaintexts/ciphertexts equations as necessary without increasing the number of variables. In the case of a hash function, the variables are related to the input message(s), which prevents the possibility to have as many equations as variables in general.

G.5 Details on Gröbner Basis Attacks

Gröbner bases are a toolkit to solve arbitrary fully determined polynomial systems. Let $\mathbb{K}$ be a field, and let $P = \mathbb{K}[x_1, \ldots, x_n]$ be the polynomial ring in n variables. For the definition of Gröbner bases one requires term orders on P to sort the monomials. Observe that a monomial $\prod_{i=1}^n x_i^{a_i}$ can be identified with its integer exponent vector $(a_1, \ldots, a_n) \in \mathbb{Z}_{\geq 0}^n$. A term order $>$ on P is then simply given by a total order on the integer exponent vectors in $\mathbb{Z}_{\geq 0}^n$. Given a non-zero polynomial $\mathsf{F} \in P$ and a term order $>$ on P, we can now sort the monomials in F under $>$. The largest monomial is then called the $>$-leading monomial of F, and we denote it as $\mathrm{LM}_>(\mathsf{F})$. Now let $I \subset P$ be an ideal, i.e. I is a multiplicatively closed additive subgroup of P. A $>$-Gröbner basis of I is a finite generating set $\mathsf{G} \subset P$ of I such that

$$(\mathrm{LM}_>(\mathsf{F}) \mid \mathsf{F} \in I) = (\mathrm{LM}_>(g) \mid g \in \mathsf{G})\,. \tag{20}$$

I.e., the leading monomials of the generating set G are minimal under $>$. For a general introduction to the theory of Gröbner bases we refer the reader to [40].

Typical choices of term orders are the *lexicographic* (LEX) and the *degree reverse lexicographic* (DRL) term orders.

Example 1. Let $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n) \in \mathbb{Z}_{\geq 0}^n$.

(1) $a >_{LEX} b$ if the leftmost non-zero entry of $a - b$ is positive.
(2) $a >_{DRL} b$ if either $\sum_{i=1}^n a_i > \sum_{i=1}^n b_i$ or $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and the rightmost entry of $a - b$ is negative.

A Gröbner basis attack on a cryptographic function, like pSquare-hash (Table 7), proceeds in three steps.

(1) Set up a fully determined polynomial model $\mathsf{F} \subset \mathbb{F}_p[x_1, \ldots, x_n]$ representing the function.
(2) Compute a Gröbner basis G of the ideal (F), for example with respect to the DRL order.
(3) Construct a univariate polynomial and factor it. Such a polynomial can always be found via term order conversion to a LEX order.

The most efficient algorithms to compute DRL Gröbner bases are Faugère's F4 [50] and Matrix-F5 [51] algorithms. These algorithms perform Gaussian elimination on matrices which represent all possible polynomial combinations of the input system F up to degree d. Their complexity is bounded by $\mathcal{O}\left(\binom{n+d}{d}^\omega\right)$, where $2 \leq \omega < 2.371552$ [115] is a linear algebra constant.

Term order conversion from a $>$-Gröbner basis to a LEX Gröbner basis can be performed via the FGLM algorithm [52, 53, 54]. In particular, for a DRL Gröbner basis $\mathsf{G} \subset P = \mathbb{F}_p[x_1, \ldots, x_n]$ whose quotient ring $P/(\mathsf{G})$ has $\mathbb{F}_p$-vector space dimension d the complexity of term order conversion from DRL to LEX is bounded by $\tilde{\mathcal{O}}\left(d^\omega\right)$ field operations (provided that the LEX basis is in shape position and that some preprocessing blocks are for free). Alternatively, univariate polynomials can be directly constructed via the characteristic polynomials of the so-called multiplication matrices [78, Chapter 6]. For example, this route was chosen in the analysis of [11, 109]. Note that the degree of these univariate polynomials is always equal to d.

Finally, the univariate polynomial $f \in \mathbb{F}_p[x]$ has to be factored. Over a finite field $\mathbb{F}_p$, it is recommended to first compute the greatest common divisor (GCD) with the field equation $g(x) = \gcd\left(f(x), x^p - x\right)$. This has the convenient benefit that all roots of f are restricted to $\mathbb{F}_p$. Let $d = \deg(f)$, if $d \leq q$, then the GCD can be computed in $\mathcal{O}\Big(d \cdot \log_2(d) \cdot \log_2\left(\log_2(d)\right) \cdot \left(\log_2(d) + \log_2(p)\right)\Big)$ field operations [12, §3.1]. Otherwise, it requires $\mathcal{O}\Big(d \cdot \log_2(d)^2 \cdot \log_2\left(\log_2(d)\right)\Big)$ field operations [91]. With $e = \deg(g)$ the final factoring step then requires $\tilde{\mathcal{O}}\big(e^2 \cdot \log(p)\big)$ field operations [35].

Typical polynomial models in cryptographic settings are high-degree systems, which work only with the input/outputs of the function, or low-degree models,

which introduce intermediate state variables after every round. For security of pSquare-hash we have to analyze the preimage problem for sponge and compression mode as well as the CICO problem for sponge mode, see Table 7 where $\mathsf{H} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ denotes a pSquare-hash instance in the respective mode. In the following paragraphs we illustrate the construction of the low-degree model for the sponge preimage problem, as well as the preprocessing step which transforms pSquare-hash polynomial models into structured multivariate quadratic polynomial systems. The preprocessing is analogous to mid-pSquare polynomial systems [9, §3.2.4].

Construction of Preimage Polynomial Systems. Let us consider a preimage problem for pSquare-hash with $n = 16$ rate $r = 8$ and R rounds. For the high-degree model, we simply translate the equation

$$\mathsf{H}\begin{pmatrix}\mathbf{x}_{in}\\ \mathtt{IV}\end{pmatrix} = \begin{pmatrix}\mathbf{h}\\ \mathbf{x}_{out}\end{pmatrix} \tag{21}$$

into 8 polynomials of degree 4^r in 8 variables, where $\mathbf{x}_{in} = (x_{in,1}, \ldots, x_{in,8})$ are the input variables, $\mathtt{IV} \in \mathbb{F}_p^8$ is the initial value, $\mathbf{h} \in \mathbb{F}_p^8$ is the hash value and $\mathbf{x}_{out} = (x_{out,1}, \ldots, x_{out,8})$ are variables representing the truncated output. Note that in practice we can ignore the 8 equations which contain $\mathbf{x}_{out}$.

For the low degree model, let $\mathsf{R}_i : \mathbb{F}_p^n \to \mathbb{F}_p^n$ denote the i^{th} round function of pSquare-hash, and for $1 \leq i \leq R-1$ let $\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_n^{(i)}\right)$ be intermediate state variables. Now we simply decompose Equation (21) into its rounds

$$\begin{aligned}
\mathsf{R}_1\begin{pmatrix}\mathbf{x}_{in}\\ \mathtt{IV}\end{pmatrix} &= \mathbf{x}^{(1)},\\
\mathsf{R}_i\left(\mathbf{x}^{(i-1)}\right) &= \mathbf{x}^{(i)}, \qquad 2 \leq i \leq R-1,\\
\mathsf{R}_R\left(\mathbf{x}^{(R-1)}\right) &= \begin{pmatrix}\mathbf{h}\\ \mathbf{x}_{out}\end{pmatrix}.
\end{aligned} \tag{22}$$

Note that this time we do not truncate the equations containing $\mathbf{x}_{out}$. Also, observe that for pSquare-hash the high degree system from Equation (21) is always contained in the ideal of the low degree one since one can simply substitute all rounds iteratively into each other.

Analogous systems can be derived for other pSquare-hash instances and modes.

Preproccesing of the Low Degree Model. Next let us illustrate that the low degree model for pSquare-hash can be reduced to a multivariate quadratic polynomial system. For this purpose, it is sufficient to consider the 4 polynomials $\mathsf{F}(\mathbf{x}) = \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are variables, representing our Feistel function $\mathsf{F} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$. Since the non-linear components share the same quartic terms, we can simply

consider their difference:

$$\begin{pmatrix} x_0 \\ x_1 \\ \left((x_1+c)^2+x_0+x_1+c'\right)^2+(x_1+c)^2+x_0+x_2 \\ \left((x_1+c)^2+x_0+x_1+c'\right)^2+2\cdot(x_1+c)^2+2\cdot x_0+x_1+x_3+c' \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{23}$$

$$\Rightarrow \begin{pmatrix} x_0 \\ x_1 \\ \left((x_1+c)^2+x_0+x_1+c'\right)^2+(x_1+c)^2+x_0+x_2 \\ (x_1+c)^2+x_0+x_1-x_2+x_3+c' \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ -y_2+y_3 \end{pmatrix}. \tag{24}$$

Next we rearrange $(x_1+c)^2 = -x_0-x_1+x_2-x_3-y_2+y_3-c'$, and substitute this expression into the third component. This yields

$$\begin{pmatrix} x_0 \\ x_1 \\ (x_2-x_3-y_2+y_3)^2 \\ (x_1+c)^2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ x_1-2\cdot x_2+x_3+2\cdot y_2-y_3+c' \\ -x_0-x_1+x_2-x_3-y_2+y_3-c' \end{pmatrix}. \tag{25}$$

So, we can reduce the quartic polynomials to two quadratic polynomials, whose quadratic terms are powers of linear polynomials.

Obviously, we can perform this transformation for every Feistel function in every round of the low degree model, which yields a multivariate quadratic polynomial system. After this preprocessing all quadratic polynomials in our model are of the form

$$\mathsf{L}_i^2 + \mathsf{A}_i = 0, \tag{26}$$

where L_i is a linear and A_i is an affine polynomial.

It was observed in [109] that for systems as in Equation (26) a change of coordinates relative to the L_i's might already yield a DRL Gröbner basis. In the following paragraphs we are going to discuss when this is indeed the case.

Sponge Mode. For simplicity, let R denote the total number of rounds, let r denote the rate and d the digest, and let $\mathcal{F} \subset \mathbb{F}_p\left[\mathbf{x}_{in}, \mathbf{x}^{(i)}, \mathbf{x}_{out} \;\middle|\; 1 \leq i \leq R-1\right]$ denote the low degree CICO/preimage polynomial system of pSquarehash. Then, the low degree models consist of $n \cdot R$ polynomials in $r + n \cdot (R-1) + \begin{cases} n-d, & \text{pre,} \\ r, & \text{CICO} \end{cases}$ many variables. For the preprocessing step, we invert the linear layer in every round, and then perform the previously outlined transformation. After the transformation we yield $\frac{n \cdot R}{2}$ affine and $\frac{n \cdot R}{2}$ quadratic polynomials. Next we perform Gaussian elimination on the affine polynomials, and eliminate $\frac{n \cdot R}{2}$ variables. Hence, we reduce to $\frac{n \cdot R}{2}$ quadratic polynomials in

$\frac{n \cdot R}{2} + r - n + \begin{cases} n-d, & \text{pre,} \\ r, & \text{CICO} \end{cases}$ variables, which have the shape of Equation (26). For simplicity, we also denote these remaining quadratic polynomials with $\mathcal{F}$.

Also, observe that for $d = r = \frac{n}{2}$ our model $\mathcal{F}$ is fully determined. However, for $n = 24$ we have $r = 16$ and $d = 8$, so $\mathcal{F}$ is underdetermined and we have 8 variables which we can choose freely.

We have implemented the pSquare-hash polynomial models in `SageMath` [114] and `OSCAR` [113] computer algebra systems. We implemented the preprocessing step as well as the change of coordinates for quadratic polynomials as in Equation (26). Moreover, in all reported experiments we performed the coordinate change to produce as many polynomials of the form of Equation (26) as possible. All Gröbner basis experiments reported in the following paragraphs have been performed on an AMD EPYC-Rome (48) CPU with 94 GB RAM.

$r = d = n$ / 2. For $n \in \{8, 12, 16, 24\}$, $r = d = \frac{n}{2}$ and the CICO/preimage problem, with our `SageMath` implementation we observed for all $5 \leq R \leq 96$ that after preprocessing the L_i's in the quadratic components (Equation (26)) are linearly independent. So, we can perform a fully determined change of coordinates to produce polynomials $\hat{x}_i^2 + \mathsf{C}_i = 0$, where C_i is affine. Under DRL our quadratic polynomials then have pairwise coprime leading monomials, so we have obtained a DRL Gröbner basis [40, Chapter 2 §9]. Moreover, the $\mathbb{F}_p$-vector space dimension of $\mathcal{F}$ is given by

$$d_{\mathcal{F}} = 2^{\frac{n \cdot R}{2}}. \tag{27}$$

$n = 24$ & $r = 16$ & $d = 8$. For the larger sponge, the success of a change of coordinates depends on the guessing strategy. Also, preimage and CICO have the same number of variables, hence we discuss the CICO problem only. We investigated the following guessing strategies.

(i) Guess $x_{in,12}, \ldots, x_{in,15}$ and $x_{out,12}, \ldots, x_{out,15}$ (CICO) or $x_{out,0}, \ldots, x_{out,3}$ (preimage) to reduce to sponge problems with rate and capacity 12.
(ii) Guess $x_0^{(1)}, \ldots, x_{\frac{n}{3}-1}^{(1)}$, which linearizes 8 squares.
(iii) Guess $x_{\frac{n}{2}-\frac{n}{6}}^{(1)}, \ldots, x_{\frac{n}{2}+\frac{n}{6}-1}^{(1)}$, which linearizes 12 squares.
(iv) Guess $x_{\frac{n}{2}-\frac{n}{6}}^{(2)}, \ldots, x_{\frac{n}{2}+\frac{n}{6}-1}^{(2)}$, which linearizes 12 squares.

For guess (i), we are in the scenario $r = d = \frac{n}{2}$ where a fully determined change of coordinates to a DRL Gröbner basis is possible. Then, we have $\mathbb{F}_p$-vector space dimension

$$d_{\mathcal{F}} = 2^{12 \cdot R}. \tag{28}$$

For guesses (ii)-(iv) a change to a DRL Gröbner basis is not possible. Therefore, we performed experimental Gröbner basis computations for the CICO system with `OSCARS`'s F4 algorithm for a downscaled version with $n = 12$ and $r = 8$. The observed highest F4 working degrees and $\mathbb{F}_p$-vector space dimensions are listed in Table 14. For increasing round numbers we observed a linear increase in

the F4 working degree, though for the $\mathbb{F}_p$-vector space dimension we observed a significant gap between the generic expected dimension $\frac{n \cdot R}{2} - \sharp$*eliminated squares* and the empirical one.

Table 14. Sponge mode CICO guessing experiments for $n = 12$ and $r = 8$.

	$x_0^{(1)}, \ldots, x_{\frac{n}{3}-1}^{(1)}$		$x_{\frac{n}{2}-\frac{n}{6}}^{(1)}, \ldots, x_{\frac{n}{2}+\frac{n}{6}-1}^{(1)}$		$x_{\frac{n}{2}-\frac{n}{6}}^{(2)}, \ldots, x_{\frac{n}{2}+\frac{n}{6}-1}^{(2)}$	
R	max. F4 deg.	$\mathbb{F}_p$-dim.	max. F4 deg.	$\mathbb{F}_p$-dim.	max. F4 deg.	$\mathbb{F}_p$-dim.
2	3	2^2	3	2^4	5	2^7
3	4	2^8	4	2^6	5	2^{10}
4	7	–	6	2^{12}	6	2^{12}
5	–	–	≥ 7	–	≥ 7	–

$n = 24$ & $r = 16$ & $d = 16$. This parameter instance is deployed in SPHINCS$^+$ (248) for message hashing, see Table 4. The CICO problem is identical to $d = 8$, hence we discuss the preimage problem only. Again, the change of coordinates to a DRL Gröbner basis is not possible, but the input system is fully determined since $r + n - d = n$.

The observed highest F4 working degrees and $\mathbb{F}_p$-vector space dimensions are listed in Table 15. Again, we observed a linear increase in the F4 working degree as well as an exponential increase in the $\mathbb{F}_p$-vector space dimension.

Table 15. Sponge mode preimage experiments with rate and digest $r = d = \frac{3}{4} \cdot n$.

	$n = 12$		$n = 24$	
R	max. F4 deg.	$\mathbb{F}_p$-dim.	max. F4 deg.	$\mathbb{F}_p$-dim.
2	5	2^8	≥ 8	–
3	8	2^{14}	–	–
4	≥ 9			

Compression Mode. Let R again denote the total number of rounds of pSquarehash, and let $\mathcal{F} \subset \mathbb{F}_p\left[\mathbf{x}_{in}, \mathbf{x}^{(i)}, \mathbf{x}_{out} \;\middle|\; 1 \leq i \leq r-1\right]$ denote the compression mode preimage polynomial system. This system consists of $n \cdot R$ polynomials in $n \cdot R + \frac{n}{2}$ variables, and after the preprocessing step we reduce to $\frac{n}{2}$ quadratic polynomials in $\frac{n}{2} \cdot (R+1)$ variables, which have the shape of Equation (26). For simplicity, we also denote these remaining quadratic polynomials with $\mathcal{F}$.

Like for the larger sponge, the success of a change of coordinates depends on the guessing strategy. For $n \in \{8, 12, 16\}$ we investigated the following guessing strategies.

(i) Guess $x_{in,\frac{n}{2}}, \ldots, x_{in,n-1}$ to reduce to a sponge with feed-forward.

Table 16. Compression mode guessing experiments.

	$x_0^{(1)}, \ldots, x_{\frac{n}{2}-1}^{(1)}$		$x_{\frac{n}{2}}^{(1)}, \ldots, x_{n-1}^{(1)}$		$x_{\frac{n}{4}}^{(1)}, \ldots, x_{\frac{3\cdot n}{4}-1}^{(1)}$	
R	max. F4 deg.	$\mathbb{F}_p$-dim.	max. F4 deg.	$\mathbb{F}_p$-dim.	max. F4 deg.	$\mathbb{F}_p$-dim.
			$n = 8$			
4	4	$5 \cdot 2^3$	2	2^{12}	5	2^8
5	6	$7^2 \cdot 2^4$	2	2^{16}	7	$7 \cdot 2^9$
6	8	$13 \cdot 2^{10}$	2	2^{20}	9	–
7	≥ 8	–	2	2^{24}	–	–
			$n = 12$			
4	5	$5 \cdot 2^7$	2	2^{18}	7	$3 \cdot 2^{12}$
5	8	–	2	2^{24}	≥ 7	–
			$n = 16$			
4	6	$5 \cdot 7 \cdot 2^7$	≥ 8	–	≥ 8	–
5	≥ 7	–	–	–		

(ii) Guess the first $\frac{n}{2}$ entries of $L_{n\times n}\mathbf{x}_{in}$ to linearize the first round.
(iii) Guess $x_0^{(1)}, \ldots, x_{\frac{n}{2}-1}^{(1)}$ to linearize the second round.
(iv) Guess $x_{\frac{n}{2}}^{(1)}, \ldots, x_{n-1}^{(1)}$ to linearize the third round.
(v) Guess $x_{\frac{n}{4}}^{(1)}, \ldots, x_{\frac{3\cdot n}{4}-1}^{(1)}$ to linearize $\frac{3}{4} \cdot n$ squares.

For guess (i), with our `SageMath` implementation we observed for all $7 \leq R \leq 96$ that a fully determined change of coordinates is possible, and the systems have $\mathbb{F}_p$-vector space dimension

$$d_{\mathcal{F}} = 2^{\frac{n \cdot R}{2}}. \tag{29}$$

For guess (ii), we observed for all $5 \leq R \leq 96$ that a fully determined change of coordinates is possible, and the system has $\mathbb{F}_p$-vector space dimension

$$d_{\mathcal{F}} = 2^{\frac{n \cdot (R-1)}{2}}. \tag{30}$$

For guesses (iii), (iv) with $n = 16$ and (v), a change to a DRL Gröbner basis is not possible, therefore we again performed experiments which are reported in Table 16. We again observed a linear increase in the F4 working degree as well as a gap for the $\mathbb{F}_p$-vector space dimension between the generic expected dimension $\frac{n \cdot r}{2} - \sharp$*eliminated squares* and the empirical one.

For guess (iv) with $n = 8, 12$, the change of coordinates is again successful and the $\mathbb{F}_p$-vector space dimension is given by Equation (30).

Security Estimation. Let us now provide a security estimation of pSquare-hash against Gröbner basis attacks.

Sponge. For sponge mode, our target security level is 124 bits. For $n = 16$, we can construct a DRL Gröbner basis with negligible effort and henceforth accurately predict the degree of the univariate polynomial in the last step of the attack. It is of degree $2^{8 \cdot R}$, so for $R = 20$ the degree already exceeds the target security level of 124 bits. For $n = 24$, we could of course pick a guess which immediately produces a Gröbner basis. But then we have $\mathbb{F}_p$-vector space dimension $2^{12 \cdot r}$, and henceforth for $R = 11$ rounds we already exceed the security level. On the other hand, for guesses, which trigger a non-trivial Gröbner basis computation, we observed a linear increase in the F4 working degree (Table 14). Recall that we have $\approx 12 \cdot R$ quadratic polynomials in $\approx 12 \cdot R$ variables. Assuming that F4's maximal degree grows with $d_{\text{F4}} \approx 4 \cdot \lfloor \frac{R}{4} \rfloor$, then for $R = 16$ already $\binom{12 \cdot R + d_{\text{F4}}}{d_{\text{F4}}}^2 \geq 2^{156}$. Likewise, if we assume that every block, consisting of 4 rounds, contributes a factor $\geq 2^{12}$ to the $\mathbb{F}_p$-vector space dimension, then 10 blocks achieve dimension $\geq 2^{120}$.

Tweaked Sponge. For tweaked sponge mode in SPHINCS$^+$ applications, the target security level is $31 \cdot d \in \{124, 186, 248\}$ against (second-)preimages, where d is the size of the digest. For $n \in \{8, 12, 16\}$ and $r = d = \frac{n}{2}$ we again have the quadratic DRL Gröbner basis, so we just have to satisfy that $2^{\frac{n \cdot R}{2}} \geq 2^{31 \cdot d}$ So, we require at least $R = 31$ rounds.

For $n = 24$, $r = 16$ and $d \in \{8, 16\}$, with the same rationale as for the untweaked sponge we have for $R = 28$ that already $\binom{12 \cdot R + d_{\text{F4}}}{d_{\text{F4}}}^2 \geq 2^{277}$. For term order conversion with FGLM and $\omega = 2$, $R = 44$ is sufficient for the complexity $\tilde{\mathcal{O}}\left(2^{\omega \cdot 132}\right)$ to exceed 248 bits.

Compression. For compression mode with $n \in \{8, 12, 16\}$ we limit our discussion to the tweaked cases with security levels $\{124, 186, 248\}$. Similar to the large sponge, we can guess variables to immediately reduce to quadratic DRL Gröbner basis with $\mathbb{F}_p$-vector space dimension $2^{\frac{n \cdot (R-1)}{2}}$. Then, $R = 32$ is sufficient to exceed the security targets. On the other hand, we can again perform a guess, which triggers a non-trivial Gröbner basis computation, but further reduces the $\mathbb{F}_p$-vector space dimension. Recall that we have $\approx \frac{n \cdot R}{2}$ quadratic polynomials in $\approx \frac{n \cdot R}{2}$ variables. In our experiments (Table 16), we observed a linear increase in the highest F4 working degree. Assuming that F4's maximal degree grows with $d_{\text{F4}} \approx 4 \cdot \lfloor \frac{R}{4} \rfloor$, then

- for $n = 8$ and $R = 20$ we have $\binom{4 \cdot R + d_{\text{F4}}}{d_{\text{F4}}}^2 \geq 2^{137}$,
- for $n = 12$ and $R = 24$ we have $\binom{6 \cdot R + d_{\text{F4}}}{d_{\text{F4}}}^2 \geq 2^{186}$, and
- for $n = 16$ and $R = 32$ we have $\binom{8 \cdot R + d_{\text{F4}}}{d_{\text{F4}}}^2 \geq 2^{282}$.

Likewise, if we assume that every block, consisting of 4 rounds, contributes a factor $\geq 2^{\frac{3 \cdot n}{4}}$ to the $\mathbb{F}_p$-vector space dimension, then for $R = 44$ we achieve

dimension $\geq \begin{Bmatrix} 2^{66}, & n = 8, \\ 2^{99}, & n = 12, \\ 2^{132}, & n = 16 \end{Bmatrix}$. Thus, for term order conversion with FGLM and $\omega = 2$ the complexities exceed the security levels $\{124, 186, 248\}$.

MitM Model for Compression. Compression mode also has a MitM polynomial model. Let $\mathsf{F}_1 : \mathbb{F}_p^n \to \mathbb{F}_p^n$ represent the first $\frac{R}{2}$ rounds of pSquare-hash, and let $\mathsf{F}_2 : \mathbb{F}_p^n \to \mathbb{F}_p^n$ represent the final $\frac{R}{2}$ many rounds. Then we have

$$\text{pSquare-hash}\left(\mathbf{x}_{in}\right) + \mathbf{x}_{in} = \begin{pmatrix} \mathbf{h} \\ \mathbf{x}_{out} \end{pmatrix}, \tag{31}$$

$$\Rightarrow \mathsf{F}_1\left(\mathbf{x}_{in}\right) = \mathbf{x}_{mid}, \qquad \mathsf{F}_2\left(\mathbf{x}_{mid}\right) + \mathbf{x}_{in} = \begin{pmatrix} \mathbf{h} \\ \mathbf{x}_{out} \end{pmatrix}. \tag{32}$$

We can rearrange Equation (32) for $\mathbf{x}_{in}$ and eliminate it, which yields

$$\mathsf{F}_1^{-1}\left(\mathbf{x}_{mid}\right) + \mathsf{F}_2\left(\mathbf{x}_{mid}\right) = \begin{pmatrix} \mathbf{h} \\ \mathbf{x}_{out} \end{pmatrix}. \tag{33}$$

This is a polynomial system consisting of n equations of degrees $\leq 4^{\frac{R}{2}}$ in $n + \frac{n}{2}$ variables. After guessing $\frac{n}{2}$ variables we can then attempt system solving techniques to recover a preimage. However, according to our interpolation attack analysis (Sections 5.2, F and G.1) the pSquare-hash permutation is dense after $\frac{R}{2}$ many rounds. Thus, the number of monomials in Equation (33) is expected to be close to the security level and the attack is nullified.

H Rebound Attacks

Rebound attacks [80,89] are one of the most effective tools on the cryptanalysis of AES-like hash functions. 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 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 `Monolith` and Skyscraper [7,8]. In the outbound phase, each solution of the inbound phase is propagated outwards in both directions.

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.

Let us asses the complexity of covering the inbound phase with polynomial system solving techniques. Let $\mathsf{S} = \mathsf{R}_{R-1} \circ \cdots \circ \mathsf{R}_0$ denote the composition of R pSquare-hash rounds. Starting point for the inbound phase is the differential equation

$$\mathsf{S}(x + \Delta_x) - \mathsf{S}(x) = \Delta_y, \tag{34}$$

where x, Δ_x and Δ_y are variables. Now we fix variables until we obtain a fully determined system. If we guess k entries in Δ_y, then Equation (34) can be viewed as function $\mathbb{F}_p^n \times \mathbb{F}_p^n \to \mathbb{F}_p^k$, and a solution to the system can be viewed as collision. Therefore, by the birthday bound the complexity of the solving Equation (34) must not exceed $\mathcal{O}\left(p^{\frac{k}{2}}\right)$.

In the extremal case, we can reduce to a univariate system by guessing x, $n-1$ entries of Δ_x and one entry of Δ_y. In addition, by clever choice of the guess in Δ_x the first round as well as the first square in the second round are inactive, see Figure 3. So, after R rounds our univariate equation has degree $2 \cdot 4^{R-2}$. Thus, as soon as $2 \cdot 4^{R-2} \geq 2^{\frac{31}{2}}$, hence $R \geq 9.25$, the complexity of finding the roots of our univariate equation exceeds the birthday bound. Based on our analysis of truncated differentials, we expect that not more than 9 rounds can be covered in one direction above probability 2^{-124} ($n = 8$) and not more than 12 rounds can be covered in one direction above probability 2^{-248} ($n \in \{16, 24\}$), see Table 6. Thus, with univariate solving we conjecture that not more than $2 \cdot 9 + 10 = 28$ rounds ($n = 8$) and $2 \cdot 12 + 10 = 34$ rounds ($n \in \{16, 24\}$) respectively can be covered.

Since the complexity of solving generic polynomial system increases with the number of variables, we do not expect that moving to multivariate solving techniques improves upon the length for the inbound phase. We therefore conjecture that pSquare-hash has sufficient security margin to resist rebound attacks.

I Unprotected Plain performance

We further provide a plain implemention of pSquare-hash within `Plonky3` [103]. Besides Mersenne-31, `Plonky3` also supports BabyBear $p = 2^{31} - 2^{27} + 1$ and KoalaBear $p = 2^{31} - 2^{24} + 1$ prime fields. Moreover, `Plonky3` has implementations of `Monolith` [60], Poseidon [61], Poseidon2 [62] and Rescue [2].

In Table 17 we compare the performance of pSquare-hash against the aforementioned primitives. Our benchmarks were performed with commit `bc7f8d4` on an INTEL(R) Core(TM) i7-10700 CPU @ 2.90GHz. Architecture specific accelerations of field arithmetic were not enabled. Over BabyBear and KoalaBear pSquare-hash was instantiated with random round constants.

`Monolith`, which utilizes non-prime field native boolean functions, is the most performant primitive for $n = 16$. All other hash functions are based on finite field native functions, among them Poseidon2 is the fastest followed by pSquare-hash.

In Table 18 we present the performance of tweaked pSquare-hash. For tweak sizes $4 \leq \tau \leq 16$ the tweaked pSquare-hash is more performant than pSquare-hash with $n = 24$. Hence, within hash-based signature applications on CPU platforms we also conclude that a smaller tweaked permutation provides better performance than a larger standard permutation.

Table 17. Plain performance of ZK-friendly hash functions in `Plonky3`. Average evaluation times for 1000 samples.

	Time (ns)					
	$n = 16$			$n = 24$		
Primitive	BabyBear	KoalaBear	Mersenne-31	BabyBear	KoalaBear	Mersenne-31
pSquare-hash	1175	1202	1035	1830	1818	1599
`Monolith`	–	–	419	–	–	–
Poseidon	1705	–	1492	5443	–	–
Poseidon2	942	793	711	1597	1261	1218
Rescue	12427	–	6751	18964	–	–

Table 18. Plain performance of tweaked pSquare-hash in `Plonky3` with $n = 16$. Average evaluation times for 1000 samples.

Tweak size	4	6	8	10	12	14	16
Time (µs)	1.268	1.288	1.350	1.452	1.462	1.533	1.594

J Proof System Performance

Next we discuss the performance of pSquare-hash for various arithmetization strategies in Zero-Knowledge (ZK) proof system.

J.1 R1CS

A Rank-1 Constraint System (R1CS) is a system of N constraints on M variables $x_0, \ldots, x_M \in \mathbb{F}_p$, with $x_0 = 0$ such that

$$\left(\sum_i \alpha_{i,j} \cdot x_i\right) \cdot \left(\sum_i \beta_{i,j} \cdot x_i\right) = \sum_i \gamma_{i,j} \cdot x_i, \qquad 0 \leq j \leq N-1, \tag{35}$$

with $\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j} \in \mathbb{F}_p$ constants for the j-th constraint. In our setting, the constraints represent the preimage relation $\mathsf{H}(\mathbf{a}) = \mathbf{b}$, and an assignment to the variables x_i serves as witness for the ZK proof. The proof system then generates an argument that the witness satisfies the constraint system without revealing the witness itself. The prover complexity scales in the number of witnesses. Moreover, in R1CS arithmetization the $\mathsf{H}(\mathbf{a}) = \mathbf{b}$ statement is translated into multiplications of linear combinations of the witness variables. In particular, linear operations in H can be absorbed into constraints for the non-linear operations. Therefore, linear operations in R1CS are considered for free, and one only needs to count the number of multiplications necesseray to represent the preimage relation.

Back to pSquare-hash, containing the number of multiplications is straightforward

$$N_{\text{pSquare-hash}} = \frac{n}{2} \cdot R, \tag{36}$$

where R denotes the total number of rounds. Since in signature settings the tweak of a pSquare-hashis a priori known, we can consider the tweak additions as constants, and henceforth the tweak schedule does not contribute to the number of R1CS constraints.

For a power permutation $x \mapsto x^d$, let d_{inc} denote the number of multiplications necessary for evaluation. Then, the number of multiplications for Poseidon and Rescue are given by

$$N_{\text{Poseidon}} = d_{inc} \cdot (n \cdot R_f + R_p), \qquad N_{\text{Rescue}} = 2 \cdot d_{inc} \cdot n \cdot R, \tag{37}$$

where R_f denotes the number of full and R_p the number of partial rounds of Poseidon and R the number of rounds for Rescue.

In Table 19 we evaluate these formulae for the primitives over 31 bit prime fields and security level 128 bits. Except for Poseidon with $d = 3$, pSquare-hash always has the least number of R1CS constraints.

Table 19. Number of R1CS constraints for pSquare-hash, Poseidon and Rescue. Numbers in parantheses are for tweaked pSquare-hash.

	Time (ns)					
	$n = 16$			$n = 24$		
Primitive	$d = 3$	$d = 5$	$d = 7$	$d = 3$	$d = 5$	$d = 7$
pSquare-hash	416 (480)			624 (720)		
Poseidon	296	426	564	430	642	852
Rescue	512	768	1024	768	1152	1546

J.2 Plonk

In Plonk [57] all constraints are expressed in terms of a quadratic polynomial

$$q_{L_i} \cdot x_i + q_{R_i} \cdot y_i + q_{O_i} \cdot z_i + q_{M_i} \cdot x_i \cdot y_i + q_{C_i} = 0, \tag{38}$$

where $q_{L_i}, q_{R_i}, q_{O_i}, q_{M_i}, q_{C_i} \in \mathbb{F}_p$ are constants depending on the relation, and x_i, y_i, z_i are constraint variables. For the representation of a preimage relation $\mathsf{H}(\mathbf{a}) = \mathbf{b}$ all arithmetic operations in H must me represented in terms of a Plonk constraint.

To count the number of Plonk constraints for pSquare-hash we analyze its components separately.

- The input/output matrix (Section 4.2) $L_{n \times n}$ requires n constraints.
- The non-linear function (Equation (13)) $\mathsf{F} \to \mathbb{F}_p^2 \to \mathbb{F}_p^2$ requires 4 constraints. So, the non-linear part of the pSquare-hash's round function requires $\frac{4}{2} \cdot n + \frac{n}{2} = 2.5 \cdot n$ constraints.
- Addition of the outputs $\mathsf{F}_{i,j}$ and the linear layer can be combined.

- Two branches have no linear-layer additions, they require 2 for the output addition constraints.
- Computation of the matrix-vector product with $C = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ requires 2 constraints.
- $\frac{n}{2} - 4$ branches have a feedback with the matrix C, so output and matrix addition can be combined in $\frac{n}{2} - 4$ constraints.
- The two rightmost branches have $\frac{n}{2} - 4$ feedback additions, so output addition and linear layer require $2 \cdot \begin{Bmatrix} 2, & n = 16, \\ 3, & n = 24 \end{Bmatrix} = 2 \cdot \frac{n}{8}$ constraints.

So, in total output addition and the linear layer require $2 + 2\frac{n}{2} - 4 + \frac{n}{4} = \frac{3}{4} \cdot n$.

– Adding a tweak of size $4 \leq \tau \leq n$ requires τ constraints.

Again, let R denote the total number of rounds and let B denote the number of blocks of pSquare-hash. Then, the number of Plonk constraints of pSquare-hash is given by

$$N_{\text{pSquare-hash}} = 2 \cdot n + \frac{13}{4} \cdot R \cdot n + \tau \cdot B. \tag{39}$$

For Plonk constraints with 3 additions, see Equation (38), generic matrix-vector multiplication for $n \times n$ matrices requires $n \cdot \left(1 + \left\lceil \frac{n-3}{2} \right\rceil\right)$ constraints. Though, Poseidon2 applies matrices with more efficient circuits. For $n \geq 12$ Poseidon2's external matrix requires just $\frac{6}{4} \cdot n + 4 \cdot \left\lfloor \frac{\frac{n}{4}-1}{2} + n \right\rfloor$ constraints, and its internal matrix requires $\left(n + 1 + \left\lceil \frac{n-3}{2} \right\rceil\right)$ constraints. So, the number of Poseidon2 and Rescue constraints is given by

$$\begin{aligned} N_{\text{Poseidon2}} = d_{inc} \cdot (n \cdot R_f + R_p) + R_p \cdot \left(n + 1 + \left\lceil \frac{n-3}{2} \right\rceil\right) \\ + (R_f + 1) \cdot \left(\frac{6}{4} \cdot n + 4 \cdot \left\lfloor \frac{\frac{n}{4} - 1}{2} \right\rfloor + n\right), \end{aligned} \tag{40}$$

$$N_{\text{Rescue}} = 2 \cdot d_{inc} \cdot n \cdot R + (R+1) \cdot n \cdot \left(1 + \left\lceil \frac{n-3}{2} \right\rceil\right). \tag{41}$$

In Table 20 we evaluated the number of Plonk constraints, pSquare-hash is always the least performative primitive.

Table 20. Number of Plonk constraints for pSquare-hash, `Monolith`, Poseidon and Rescue. Numbers in parantheses are for tweaked pSquare-hashwith $\tau = 16$.

	Time (ns)					
	$n = 16$			$n = 24$		
Primitive	$d = 3$	$d = 5$	$d = 7$	$d = 3$	$d = 5$	$d = 7$
pSquare-hash	2736 (3392)			4104 (5088)		
Poseidon2	1172	1158	1272	1870	2046	2220
Rescue	1664	1920	2176	3360	3744	4128

J.3 AIR

The arithmetic intermediate representation used in STARK proofs is inherently related to PLONK. Similarly, the prover and the verifier agree on a relation to be proven and compute computation traces as a witness matrix. Given an agreed-upon structure of this matrix, the parties agree on a set of constraint polynomials $P_i(\boldsymbol{x})$ that evaluate to zero for a predefined subset of row indices in the matrix. Given this simple structure, it is possible to build permutation arguments, equality constraints, linear and non-linear relation constraints. However, as arithmetization is only the first step, the concrete choice of constraints and matrix arrangement has a concrete performance impact on the subsequent algorithms in a stark proof and verification. Some key decisions that govern our implementation are a balance in rows and columns of the trace matrix and a small maximum degree. Given a matrix with many rows, constructing each polynomial from the columns using the number theoretic transform gets more expensive. However, when using randomized polynomials (to prove the relation in zero-knowledge), the polynomials are extended to a minimum length, so having many very short columns also has adverse effects. In line with the implementation of Poseidon2 in Plonky3, we opted to optimize batch verification of hash evaluations. A hash preimage relation in our implementation is represented in a single row. While inefficient for individual relations, this structure amortizes well for many hashes. Next to the matrix shape, the maximum degree significantly impacts the commitment scheme. In principle, the performance of FRI scales exponentially with the maximum degree of the constraints. However, higher degree constraints allow for fewer witnesses in the matrix and a smaller proof size and verification time. In our implementation, we provide two options. First, we give a version with an intermediate constraint within the non-linear function and a resulting maximum degree of 2. Second, we omit this intermediate constraint, roughly halving our witness size but doubling our degree.

In Table 21, we present and compare the performance numbers of proving and verifying pSquare-hash preimage relations with those of Poseidon2. We divide the results into three categories. First is the witness matrix generation phase, which depends primarily on the number of elements in the table. Second is the proof generation, whose commitment phase is more heavily penalized by a large maximum constraint degree. Third is the verification, where the proof size dominates the runtime.

Overall, both versions, with degrees 4 and 2, present a reasonable trade-off for pSquare-hash, depending on the focus on fast prover or verification time. Further, we see that the amortization trick bears fruit for proof generation and verification, achieving substantial effects already with 2^{12} hashes in parallel. Finally, however, Poseidon2 seems a bit more efficient in most settings. A potential answer is that the fewer rounds lead to a smaller trace size, compensating for the higher maximum degree of Poseidon2. In addition, Poseidon2 has dedicated implementations for each supported prime in Plonky3; it remains an open question whether such dedicated optimizations could also be employed to optimize pSquare-hash. Finally, in cases where hash preimages are not a standalone

application but built into a larger proof, it might still be favorable to integrate pSquare-hash, so as not to increase the maximum degree of the total proof.

Table 21. Concrete performance of trace matrix and proof generation as well as verification of hash pre-images. Single row evaluation proves 4 pre-images in a single matrix row, while amortized proves 4096 hashes in 1024 rows. Performance is given in Throughput as elements per second. The degree parameter denotes the maximum constraint degree.

Primitive	Degree	Traces		Proof		Verify	
		Single	Amortized	Single	Amortized	Single	Amortized
Poseidon2	7	2.37e+05	2.28e+05	23.3	1.72e+03	493	2.75e+05
	3	2.23e+05	2.04e+05	22.5	2.32e+03	351	2.34e+05
pSquare-hash	4	2.03e+05	1.78e+05	23.2	1.05e+03	254	1.82e+05
	2	2.07e+05	1.53e+05	80.3	1.76e+03	148	1.28e+05

K Round-Based Hardware Implementation Comparison

Table 22. Cost and performance comparison of round-based pipelined (masked) hardware implementations of permutation functions (Keccak-f, `SKINNY-hash` F, `Ascon-p` and Poseidon2) with d shares. Evaluated in NanGate 15 nm at typical conditions.

Function	State [bits]	Tweak [bits]	d	Area [kGE]	Power [mW]	Crit. Path [ns]	Latency [cyc/perm]	Random [bits/cyc]
Keccak-f	1600	0	1	29.2	1.2932	0.3058	24/1	0
			2	210.1	9.5685	0.0785	48/2	3200
			3	424.6	19.2688	0.0817	48/2	9600
			4	712.6	32.3549	0.0844	48/2	19200
`SKINNY-Hash` F	256	0	1	6.8	1.1123	0.1286	48/1	0
			2	117.0	27.2144	0.1568	432/9	128
			3	189.2	43.2919	0.1811	432/9	384
			4	275.2	61.7962	0.1889	432/9	768
	384	0	1	13.5	2.1912	0.1302	56/1	0
			2	181.4	42.2872	0.1744	504/9	128
			3	293.3	67.1771	0.1902	504/9	384
			4	426.7	95.7838	0.1823	504/9	768
`Ascon-p`	320	0	1	10.4	3.6759	0.0814	12/1	0
			2	92.3	20.1676	0.1413	24/2	320
			3	158.7	39.7225	0.1869	24/2	960
			4	264.3	67.8261	0.1955	24/2	1920
Poseidon2$^{\pi}$	496	0	1	252.4	9.5691	7.4014	22/1	0
			2	1004.1	46.2870	6.4696	66/3	2976
			3	2232.9	105.3515	6.7447	66/3	7440
			4	3779.2	177.9558	7.0916	66/3	16864
	744	0	1	372.1	13.8153	9.7463	22/1	0
			2	1509.4	69.5717	8.7761	66/3	4464
			3	3353.8	158.7248	9.0514	66/3	11160
			4	5675.1	267.4399	9.3720	66/3	25296

Table 23. Cost and performance comparison of round-based (masked) hardware implementations of different pSquare-hash variants with d shares. Evaluated in NanGate 15 nm at typical conditions.

Function	State [bits]	Tweak [bits]	d	Area [kGE]	Power [mW]	Crit. Path [ns]	Latency [cyc/perm]	Random [bits/cyc]
pSquare-hash	248	0	1	29.6	0.9795	3.7834	52/1	0
			2	93.8	3.7497	3.4118	104/2	248
			3	179.9	8.5560	3.4231	104/2	620
			4	261.3	12.9352	3.4016	104/2	1488
		248	1	40.4	1.2313	4.1313	60/1	0
			2	113.9	5.2134	3.7002	120/2	248
			3	199.1	10.7667	3.6812	120/2	620
			4	290.4	14.0941	3.7121	120/2	1488
	372	0	1	44.6	1.4779	3.6912	52/1	0
			2	141.6	5.9388	3.4323	104/2	372
			3	270.8	12.8925	3.4333	104/2	930
			4	452.9	20.2152	3.4439	104/2	2232
		372	1	58.1	2.1313	4.2676	60/1	0
			2	165.9	7.9121	3.7220	120/2	372
			3	300.1	15.0016	3.7199	120/2	930
			4	485.2	22.6867	3.7356	120/2	2232
	496	0	1	59.2	1.9589	3.7920	52/1	0
			2	187.5	7.8744	3.4429	104/2	496
			3	377.9	17.1139	3.4443	104/2	1240
			4	535.7	27.1640	3.4968	104/2	2976
		496	1	78.1	2.4514	4.2280	60/1	0
			2	206.2	8.4404	3.7382	120/2	496
			3	396.7	17.7460	3.7373	120/2	1240
			4	615.1	27.9528	3.7473	120/2	2976
	744	0	1	89.2	2.9578	3.7760	52/1	0
			2	282.4	11.8707	3.4508	104/2	744
			3	568.6	25.7455	3.4455	104/2	1860
			4	896.7	40.8305	3.4996	104/2	4464
		744	1	117.4	3.7184	4.2039	60/1	0
			2	310.2	12.6960	3.7453	120/2	744
			3	596.3	26.6668	3.7337	120/2	1860
			4	924.3	42.0218	3.7491	120/2	4464

L Sensitivity Analysis

Table 24. Simplified sensitivity analysis of SLH-DSA's internal functions to Side-Channel Analysis (SCA) and Fault Injection Analysis (FIA) attacks.

Attack	Target	Algorithm	**F**	**PRF**	**PRF**$_{msg}$	**H**	**H**$_{msg}$	**T**$_l$
SCA	Confidentiality	Gen	++	++	n/a	−	n/a	−
		Sign	++	++	++	−	−	−
		Vrfy	−	n/a	n/a	−	−	−
FIA	Integrity	Gen	+	+	n/a	+	n/a	+
		Sign	++	++	+	++	+	++
		Vrfy	+	n/a	n/a	+	+	+

Sensitivity: High (++), Low (+), None (−), Not Applicable (n/a)

M Security Analysis of TS: Proof of Theorem 1

In this section we present the proof of the indifferentiability bound of the TS construction. We begin by introducing some useful notation and defining the simulator. Then, we introduce an intermediate world, decompose the distance by the triangle inequality and bound each component separately.

Setup. The simulator, Sim, stores its primitive queries in a table $\mathcal{C}^{\mathsf{P}}$, which comprises tuples of the form (t, X, Y, dir), where t denotes the tweak. The simulator can be accessed through two interfaces, S_{P} and $\mathsf{S}_{\mathsf{P}^{-1}}$, each simulating P and P^{-1}. The image of X by S_{P} is defined to be Y, and the query was made in the direction $\mathrm{dir} \in \{\mathrm{fwd}, \mathrm{inv}\}$. In the remaining of the proof we abuse notation by omitting the tweak when useful.

Graph Notation. From its table $\mathcal{C}^{\mathsf{P}}$, the simulator derives a tree construction. The nodes are elements of $\mathbb{F}_p^n$, representing the internal state of TS. Two nodes S_1 and S_2 are connected by an edge, with label $m \in \mathbb{F}_p^r$, if $B = \mathsf{TS}(A) \oplus (m \| 0^c)$. The label m may be empty, which can indicate squeezing. Additionally, nodes with value IV are called root nodes. Given $X, Y \in \mathbb{F}_p^n$, $m_p \in (\mathbb{F}_p)^{\leq r-1} \setminus \{\emptyset\}$ and $m_f \in \mathbb{F}_p^r$ we define four kinds of edges:

- $X \xrightarrow{m_f/a} Y$ denotes that $(X \oplus (m_f \| 0^c), Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}$;
- $X \xrightarrow{m_p/p} Y$ denotes that $(X \oplus (m_p \| 10^*), Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}$;
- $X \xrightarrow{m_f/f} Y$ denotes that $(X \oplus (m_f \| 0^c \oplus C), Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}$.
- $X \longrightarrow Y$ is a special case of the above, and it denotes that $(X, Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}$;

The symbol a stands for absorb, while p and f indicate a final absorbing call with respectively a partial or a full block. Finally, C represents the NCP constant

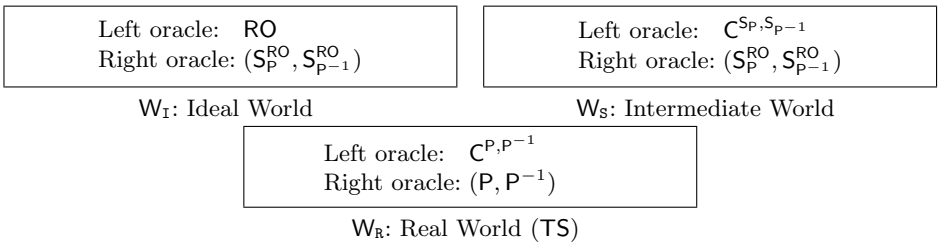


Figure 9. Worlds involved in the security proof.

used by TS. Furthermore, note that the first kind of edges correspond to absorb calls, while the last one corresponds to a squeeze call.

The set of *absorbing paths* are the nodes that correspond with *absorbing* in the real world. These paths are of the form:

$$IV \xrightarrow{m_1/a} S_2 \xrightarrow{m_2/a} \cdots \xrightarrow{m_n/a} S_{n+1}\,.$$

We will abbreviate those as $IV \xrightarrow{m_1\|\cdots\|m_n/a} S_{n+1}$. Naturally, an absorbing path is followed by a squeezing path:

$$IV \xrightarrow{m_1/a} S_2 \xrightarrow{m_2/a} \cdots \xrightarrow{m_{n-1}/a} S_m \xrightarrow{m_n/\delta} Z_1 \longrightarrow Z_2 \longrightarrow \cdots \longrightarrow Z_\ell\,,$$

for $\delta \in \{p, f\}$. We will abbreviate this as $IV \xrightarrow{m_1\|\cdots\|m_n,\ell} Z$. Note that the $/\delta$ symbol in the path is not included. This is because the message blocks are not padded in this graph representation, so that the length of $m_1\|\cdots\|m_n$ provides sufficient information to deduce whether the last call uses a full or partial block. The set of paths that are *valid* corresponds to the union of absorbing paths with squeezing paths. We make a clear distinction between them, since the simulator only needs to guarantee consistency for squeezing paths. Moreover, we define $\mathsf{Rooted}(\mathcal{C}^{\mathsf{P}})$ as the set of rooted nodes $Z \in \mathbb{F}_p^n$ from which one can build a valid path $IV \xrightarrow{M/a} Z$ or $IV \xrightarrow{M,\ell} Z$. For the sake of the proof, we introduce a table $\widehat{\mathcal{C}^{\mathsf{P}}}$, that comprises tuples of the form (X, Y, dir). With these extended tables, we can define $\mathsf{Rooted}_D(\widehat{\mathcal{C}^{\mathsf{P}}})$ as the rooted nodes reached solely through distinguisher queries.

The Simulator. The simulator keeps track of the graph construction and uses it to output consistent answers. It is essentially the same sort of simulator as in the sponge construction [21], but it manages a more complex set of valid paths. A full description of the simulator can be found in Algorithm 3.

World Splitting. Similarly to [95], we introduce an intermediate world $\mathsf{W}_{\mathtt{S}}$, as illustrated in fig. 9. In this world, the construction oracle gives access to TS,

based on the simulator, which is itself based on a random oracle hidden from the adversary.

$$\mathbf{Adv}^{\mathrm{iff}}_{\mathsf{TS},\mathsf{Sim}}(\mathcal{A}) \leq \Delta_{\mathcal{A}}\left(\mathsf{W}_{\mathtt{R}}\,;\,\mathsf{W}_{\mathtt{S}}\right) \tag{42}$$

$$+\ \Delta_{\mathcal{A}}\left(\mathsf{W}_{\mathtt{S}}\,;\,\mathsf{W}_{\mathtt{I}}\right)\,. \tag{43}$$

The bounding (42), is straight-forward: it boils down to evaluating the distance between the simulator and a random permutation. This is because the output S_{P} is always uniformly random, (c.f. Algorithm 3). This distance can thus be upper bounded by the PRP/PRF switching lemma [14] yielding:

$$\Delta_{\mathcal{A}}\left(\mathsf{W}_{\mathtt{R}}\,;\,\mathsf{W}_{\mathtt{S}}\right) \leq \frac{\binom{Q}{2}}{p^n}\,. \tag{44}$$

The remainder of the proof is dedicated to the analysis of (43).

Bad Events. Let Q denote the total number of queries made by the distinguisher and let $Q_{\mathcal{T}}$ be the maximum number of queries made with the same tweak. For $i \in [Q]$, let $\mathcal{C}^{\mathsf{P}}[i]$ denote the state of $\mathcal{C}^{\mathsf{P}}$ after the first i queries. We define a family of bad events, indexed by a query index $i \in [Q]$, and defined over $\mathcal{C}^{\mathsf{P}}$.

- **CollP**$[i]$: $\exists (X, Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}[i] \setminus \mathcal{C}^{\mathsf{P}}[i-1]$, $(X', Y', \mathrm{dir}') \in \mathcal{C}^{\mathsf{P}}[i-1]$ such that

$$\{\lfloor Z \rfloor^c, \lfloor Z \rfloor^c \oplus C\} \cap \{\lfloor Z' \rfloor^c, \lfloor Z' \rfloor^c \oplus C\} \neq \emptyset\,,$$

  where

$$(Z, Z') = \begin{cases} (Y, Y') & \text{if } \mathrm{dir} = \mathrm{fwd}\,, \\ (X, X') & \text{otherwise}\,; \end{cases}$$

- **ConnectP**$[i]$: $\exists (X, Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}[i] \setminus \mathcal{C}^{\mathsf{P}}[i-1]$, $(X', Y', \mathrm{dir}') \in \mathcal{C}^{\mathsf{P}}[i-1]$ such that

$$\{\lfloor Z \rfloor^c, \lfloor Z \rfloor^c \oplus C\} \cap \{\lfloor Z' \rfloor^c, \lfloor Z' \rfloor^c \oplus C\} \neq \emptyset\,,$$

  where

$$(Z, Z') = \begin{cases} (Y, X') & \text{if } \mathrm{dir} = \mathrm{fwd}\,, \\ (X, Y') & \text{otherwise}\,; \end{cases}$$

- **ConnectIV**$[i]$: $\exists (X, Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}[i] \setminus \mathcal{C}^{\mathsf{P}}[i-1]$, $(X', Y', \mathrm{dir}') \in \mathcal{C}^{\mathsf{P}}[i-1]$ such that

$$\{\lfloor Z \rfloor^c, \lfloor Z \rfloor^c \oplus C\} \cap \{\lfloor \mathtt{IV} \rfloor^c, \lfloor \mathtt{IV} \rfloor^c \oplus C\} \neq \emptyset\,,$$

  where

$$Z = \begin{cases} Y & \text{if } \mathrm{dir} = \mathrm{fwd}\,, \\ X & \text{otherwise}\,; \end{cases}$$

– **Guess**$[i]$: $\exists(X, Y, \mathrm{dir}) \in \mathcal{C}^{\mathsf{P}}[i] \setminus \mathcal{C}^{\mathsf{P}}[i-1]$; such that $Y \notin \mathsf{Rooted}_D(\widehat{\mathcal{C}^{\mathsf{P}}}[i])$, but $Y \in \mathsf{Rooted}(\widehat{\mathcal{C}^{\mathsf{P}}}[i])$.

We define $\mathbf{Bad}[i]$:=$\mathbf{CollP}[i] \vee \mathbf{ConnectP}[i] \vee \mathbf{ConnectIV}[i] \vee \mathbf{Guess}[i]$ and

$$\mathbf{Bad} := \bigvee_{i=1}^{Q} \mathbf{Bad}[i] \,.$$

The bad events ensure that the simulator responds consistently with respect to the random oracle. Intuitively, **CollP** addresses collisions in the inner part of the construction, modulo the xoring of the domain separating constant, ensuring that each simulator query is mapped to at most one squeezing path. While **ConnectP** and **ConnectIV** address the cases where an unrooted node becomes rooted, which can cause a discrepancy the simulator cannot resolve. Finally, notice that the simulator in $\mathsf{W_S}$ (and $\mathsf{W_R}$) has more information than that of $\mathsf{W_I}$ and **Guess** encodes this.

$\mathsf{W_S}$ *Versus* $\mathsf{W_I}$ *as Long as No Bad.* We will show that, conditioned on $\neg\mathbf{Bad}$, the worlds $\mathsf{W_I}$ and $\mathsf{W_S}$ are identically distributed. In particular, we have:

Lemma 1. *We have* $\mathsf{TS}(m) = \mathsf{RO}(m)$ *so long as* **Bad** *does not happen, and additionally*

$$\Pr\left[\mathcal{A}^{\mathsf{W_I}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right] = \Pr\left[\mathcal{A}^{\mathsf{W_S}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right] .$$

The proof of Lemma 1 can be found, mutatis mutandis, in [82, Lemmas 3 and 4].

Probability of Bad. What remains is to bound the probability of the bad events, which we do in Lemma 2.

Lemma 2. *For* $\mathsf{X} \in \{\mathsf{S}, \mathsf{I}\}$*, we have*

$$\Pr\left[\mathcal{A}^{\mathsf{W_X}} \textit{ sets } \mathbf{Bad}\right] \leq \frac{9QQ_{\mathcal{T}}}{p^c} + \frac{3Q}{p^c} \,.$$

Proof. The bad events are defined over the simulator table $\mathcal{C}^{\mathsf{P}}$, and **Guess** can only be set in $\mathsf{W_S}$. Moreover, the simulator receives extra queries in $\mathsf{W_S}$, which only increases its success probability to set **Bad**. We will thus evaluate the probability of **Bad** in $\mathsf{W_S}$. A "query" here refers to a call to the simulator, regardless whether this comes from the adversary or the construction. We have

$$\begin{aligned} \Pr_{\mathsf{W_S}}[\mathbf{Bad}] &\leq \sum_{i=1}^{Q} \Pr[\mathbf{Bad}[i] \mid \neg\mathbf{Bad}[i-1]] \\ &\leq \sum_{i=1}^{Q} \Pr[\mathbf{CollP}[i] \mid \neg\mathbf{Bad}[i-1]] \end{aligned} \tag{45}$$

$$+ \Pr[\mathbf{ConnectP}[i] \mid \neg\mathbf{Bad}[i-1]] \tag{46}$$
$$+ \Pr[\mathbf{ConnectIV}[i] \mid \neg\mathbf{Bad}[i-1]] \tag{47}$$
$$+ \Pr[\mathbf{Guess}[i] \mid \neg\mathbf{Bad}[i-1]]\,, \tag{48}$$

where $\mathbf{Bad}[0]$ denotes an event that never holds. We will evaluate (45) and (48) individually. First of all, we remark that by Lemma 1 as long as $\neg\mathbf{Bad}$ holds, the simulator samples its answers uniformly at random. Denote (t, X, Y, dir), the i-th query.

CollP$[i]$. This bad event concerns collisions over c field elements. Denote 1_t^j the variable that is set to 1 if the j-th query uses the tweak t, and 0 otherwise. For each previous query (X', Y'), the bad event covers 3 different collision scenarios involving previous queries. Regarding the first type, this only triggers the bad event if both queries share the same tweak, yielding:

$$\Pr_{\mathsf{W_s}}[\mathbf{CollP}[i] \mid \neg\mathbf{Bad}[i-1]] \leq 1_t^j \cdot \frac{3(i-1)}{p^c}\,. \tag{49}$$

ConnectP$[i]$. Similarly, we have

$$\Pr_{\mathsf{W_s}}[\mathbf{ConnectP}[i] \mid \neg\mathbf{Bad}[i-1]] \leq 1_t^j \cdot \frac{3(i-1)}{p^c}\,. \tag{50}$$

ConnectIV$[i]$. Clearly, we have

$$\Pr_{\mathsf{W_s}}[\mathbf{ConnectP}[i] \mid \neg\mathbf{Bad}[i-1]] \leq \frac{3}{p^c}\,. \tag{51}$$

Guess$[i]$. We can bound this event in the same we bounded **CollP**$[i]$: Therefore, we get:

$$\Pr_{\mathsf{W_s}}[\mathbf{Guess}[i] \mid \neg\mathbf{Bad}[i-1]] \leq 1_t^j \cdot \frac{3(i-1)}{p^c} \tag{52}$$

Then by combining (45) to (47) and (49) to (52) we get:

$$\begin{aligned}\Pr_{\mathsf{W_s}}[\mathbf{Bad}] &\leq \sum_{i=1}^{Q} \Pr[\mathbf{Bad}[i] \mid \neg\mathbf{Bad}[i-1]] \\ &\leq 3\left(1_t^j \cdot \frac{3(i-1)}{p^c}\right) + \frac{3}{p^c} \leq \frac{9QQ_{\mathcal{T}}}{p^c} + \frac{3Q}{p^c}\,.\end{aligned}$$

Conclusion. Combining (49) and (52), completes the proof of Lemma 2. □

Algorithm 3 Simulator interfaces for TS

Function $\mathsf{S}_{\mathsf{P}^{-1}}(t, Y)$

if $\exists(t, X, Y) \in \mathsf{Sim}.\mathcal{C}^{\mathsf{P}}$ **then**
  return Y
end if
$X \xleftarrow{\$} \mathbb{F}_p^n$
$\mathsf{Sim}.\mathcal{C}^{\mathsf{P}} \leftarrow \mathsf{Sim}.\mathcal{C}^{\mathsf{P}} \cup \{(X, Y)\}$
return X

Function $\mathsf{S}_{\mathsf{P}}(t, X)$

if $\exists(t, X, Y) \in \mathsf{Sim}.\mathcal{C}^{\mathsf{P}}$ **then**
  return X
end if
$Y \xleftarrow{\$} \mathbb{F}_p^n$
// Start of squeezing phase with partial block
if $\exists S \in \mathbb{F}_p^n$, $M \in \mathbb{F}_p^* \setminus \{\emptyset\}$, and $m \in \mathbb{F}_p^{\leq r-1} \setminus \{\emptyset\}$ such that $IV \xrightarrow{M/a} S$ and $X = S \oplus m \| 10^*$ **then**
  $Y_o \leftarrow \mathsf{RO}(IV, M\|m)[r]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^c$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase with full block
if $\exists S \in \mathbb{F}_p^n, M \in \mathbb{F}_p^* \setminus \{\emptyset\}$, and $m \in \mathbb{F}_p^r$ such that $IV \xrightarrow{M/a} S$ and $X = S \oplus m \| 0^c \oplus C$ **then**
  $Y_o \leftarrow \mathsf{RO}(IV, M\|m)[r]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^c$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// In the middle of a squeezing phase
if $\exists M \in \mathbb{F}_p^*$, and $\ell \geq 1$ such that $[IV] \xrightarrow{M,\ell} X$ **then**
  $Y_o \leftarrow \mathsf{RO}(IV, M)[\ell r : (\ell + 1)r - 1]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^c$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase from `IV` with one partial block
if $\exists m \in \mathbb{F}_p^{\leq r-1}$ such that $X = IV \oplus m \| 10^*$ **then**
  $Y_o := \mathsf{RO}(IV, m)[r]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^c$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase from `IV` with one full block
if $\exists m \in \mathbb{F}_p^r$ such that $X = IV \oplus m \| 0^{c_I} \oplus C$ **then**
  $Y_o := \mathsf{RO}(IV, m)[r]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^c$
  $Y \leftarrow Y_o \parallel Y_i$
end if
$\mathsf{Sim}.\mathcal{C}^{\mathsf{P}} \leftarrow \mathsf{Sim}.\mathcal{C}^{\mathsf{P}} \cup \{(X, Y)\}$
return Y