Poseidon: A New Hash Function for Zero-Knowledge Proof Systems

Lorenzo Grassi, *Radboud University Nijmegen;* Dmitry Khovratovich, *Ethereum Foundation and Dusk Network;* Christian Rechberger, *IAIK, Graz University of Technology;* Arnab Roy, *University of Klagenfurt;* Markus Schofnegger, *IAIK, Graz University of Technology*

https://www.usenix.org/conference/usenixsecurity21/presentation/grassi

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

POSEIDON: A New Hash Function for Zero-Knowledge Proof Systems

Lorenzo Grassi[1], Dmitry Khovratovich[2], Christian Rechberger[3], Arnab Roy[4], and Markus Schofnegger[3]

[1]Radboud University Nijmegen
[2]Ethereum Foundation and Dusk Network
[3]IAIK, Graz University of Technology
[4]University of Klagenfurt
l.grassi@cs.ru.nl, khovratovich@gmail.com, firstname.lastname@iaik.tugraz.at, arnab.roy@aau.at

Abstract

The area of practical computational integrity proof systems, like SNARKs, STARKs, Bulletproofs, is seeing a very dynamic development with several constructions having appeared recently with improved properties and relaxed setup requirements. Many use cases of such systems involve, often as their most expensive part, proving the knowledge of a preimage under a certain cryptographic hash function, which is expressed as a circuit over a large prime field. A notable example is a zero-knowledge proof of coin ownership in the Zcash cryptocurrency, where the inadequacy of the SHA-256 hash function for such a circuit caused a huge computational penalty.

In this paper, we present a modular framework and concrete instances of cryptographic hash functions which work natively with GF(p) objects. Our hash function POSEIDON uses up to 8x fewer constraints per message bit than Pedersen Hash.

Our construction is not only expressed compactly as a circuit, but can also be tailored for various proof systems using specially crafted polynomials, thus bringing another boost in performance. We demonstrate this by implementing a 1-out-of-a-billion membership proof with Merkle trees in less than a second by using Bulletproofs.

Contents

1 Introduction

The recent advances in computational integrity proof systems made a number of computational tasks verifiable in short time and/or in zero knowledge. Several protocols appeared that require one party to prove the knowledge of a seed-derived secret, of an element being part of a large set, or their combination. Whereas accumulator-based solutions [20, 21] and algebraic Schnorr proofs exist in the area, they are quite involving and thus error-prone, require a trusted setup, are limited in statement language, and are often slow. An alternative is to express secret derivation using cryptographic hash functions, and to prove set membership by presenting an opening in a properly chosen Merkle tree, also built on a cryptographic hash function. Such hash-based protocols require a computational integrity proof system, which can be applied to an arbitrary arithmetic circuit. However, for the protocol to be

efficient, proofs must be generated and verified in reasonable time, which in turn requires the hash function to be cheap in a certain metric depending on the proof system.

In the middle of 2020, the most popular proof systems are ZK-SNARKs (Pinocchio [49], Groth16 [35], PLONK [27], Marlin [23] to name a few), Bulletproofs [19], ZK-STARKs [9], and MPC-in-the-head systems [7, 22, 29]. The former two groups have already been applied to a number of real-world protocols, whereas the latter ones are the most promising from the perspective of post-quantum security. These systems use two quite different circuit descriptions so that the proof size and generation time are computed differently:

- The R1CS format (rank-1 quadratic constraints) describes the circuit as a set of special quadratic polynomials of the form $L_1(X) \cdot L_2(X) = L_3(X)$, where X is the tuple of internal and input variables, L_i are affine forms and $\cdot$ is the field multiplication, and (possibly in an affine-equivalent form) is used in almost all SNARKs and Bulletproofs. The circuit multiplication and addition gates are defined over a prime field $\mathrm{GF}(p)$. The proof generation complexity is directly proportional to the number T of constraints, which often corresponds to the number of multiplication gates. The prime field $\mathrm{GF}(p)$ is the scalar field of an elliptic curve, where for ZK-SNARKs the curve should be pairing-friendly and for Bulletproofs it should just be a secure curve.
- The AET metric is used in ZK-STARKs and (to some extent) in the PLONK proof system. The computation is expressed as a set of internal program states related to each other by polynomial equations of degree d. The state consists of w field elements and undergoes T transformations. The proof generation is roughly proportional to the product $w \cdot d \cdot T$. The number and sparsity of polynomial constraints do not play a major role.

Our goal was to design a family of hash functions that are optimal in the R1CS (as the most widespread) and good in the AET metric, while also supporting different finite field sizes. It turned out that the substitution-permutation network (SPN) design, well-known in symmetric cryptography, allows for a generic hash function framework where the only security-critical parameter that has to be changed for each instance is the number of rounds, and we provide an efficient and transparent strategy for its choice. The S-box is chosen as the power map $x \mapsto x^d$, where $d \geq 3$ is usually chosen as the smallest integer that guarantees invertibility and provides non-linearity. In particular, the cube function x^3 is almost universally chosen, apart from cases of fields where this function is not a bijection. Instead, we suggest other S-boxes such as x^5 or $1/x$ for these cases. Thanks to a succinct representation of the functions and a low S-box degree, we are able to optimize the circuit significantly for PLONK and RedShift proof systems, with performance improvements by a factor of up to 40.

Our Contributions. We design and analyze a family of hash functions over $\mathrm{GF}(p)$ named POSEIDON. The internal permutation is called POSEIDON$^\pi$ and is based on the HADES design strategy [31], which is essentially a strategy based on substitution-permutation networks with t cells, but including the use of so-called *partial* rounds, which use non-linear functions only for part of the state. In our specific construction, only one S-box is used in these partial rounds, while full non-linear layers (i.e., t S-boxes) are used in all other rounds. This is done to reduce the R1CS or AET cost.

We aim to support security levels of 80, 128, and 256 bits, where the security is the same for collision and preimage resistance. For each pair (basic field, security level) we suggest a concrete instance of POSEIDON. In our hash function, a few S-box elements are reserved for the capacity (roughly double the security level in bits), and the rest for the rate. The permutation width is determined by the application: It is set close to 1280 bits for long-message hashing, whereas for Merkle trees we support various widths to enable 2:1, 4:1, and other arities and thus higher ZK performance.

We provide an extensive cryptanalysis of POSEIDON with an accent on algebraic methods as these prove to be the most effective. We explore different variants of interpolation, Gröbner basis, and higher-order differential attacks. As our permutations are quite wide, we do not aim for them behaving like randomly chosen permutations. Instead, for a security level of M bits we require that no attack could exhibit a non-random (but relevant for collision/preimage search) property of a permutation faster than in 2^M queries. We then calculate the maximum number of rounds for each field, security level, and fixed permutation width that can be attacked. Then we select the number of rounds for concrete instances together with a security margin.

We have evaluated the number of constraints in POSEIDON instances for the R1CS metric and the AET metric. Our primary proposals POSEIDON-80/128/256 are listed in Table 1 (BLS being BLS12-381[1], BN being BN254 [52], Ed being the Ristretto group[2]) and are compared to similar-purpose designs. Finally, we refer to [30, Appendix A] for a complete overview of our auxiliary files, including reference implementations and scripts to create POSEIDON$^\pi$ instances.

We also have third-party benchmarks of POSEIDON for regular hashing[3] (Table 1) and in ZK proof systems: PLONK (Table 6), Groth16 (Table 3), and Bulletproofs (Table 5).

[1] https://electriccoin.co/blog/new-snark-curve/
[2] https://ristretto.group
[3] https://github.com/shamatar/poseidon_hash and https://github.com/shamatar/rescue_hash

Table 1: Our primary proposals and their competitors. "Tree" refers to the Merkle tree arity and is equal to the rate/capacity ratio. "Curve" denotes the curve (BLS12-381, BN254, Ed25519) whose (subgroup) scalar field determines the prime size. The R1CS/bit costs are obtained by dividing the R1CS prover costs by the message rate. Timings are from a third-party implementation of *Rescue* and POSEIDON on an i9-8950 CPU @2.9 Ghz and 32 GB RAM.

Name	**S-box**	**Rate** bits/perm.	**SB size** ($\log_2 p$)	**Tree** arity	$\mathbf{R_F}$	$\mathbf{R_P}$	**Curve** Scalar field	**R1CS** /perm.	**R1CS** /bit	**Time** /perm.
POSEIDON-80	x^5	510	255	2:1	8	33	BLS/BN/Ed	171	0.34	0.021 ms
	x^5	1020	255	4:1	8	35		225	0.22	0.05 ms
POSEIDON-128	x^5	510	255	2:1	8	57	BLS/BN/Ed	243	0.47	0.033 ms
	x^5	1020	255	4:1	8	60		300	0.29	0.08 ms
	x^5	2040	255	8:1	8	63		405	0.2	0.259 ms
POSEIDON-256	x^5	1020	255	2:1	8	120	BLS/BN/Ed	504	0.5	0.216 ms
	x^5	2040	255	4:1	8	120		600	0.3	0.578 ms
Pedersen Hash	-	516	-	2:1	-		BLS12-381	869	1.68	
Rescue	x^5 & $x^{1/5}$	510	255	2:1	16		BLS/BN/Ed	268	0.52	0.525 ms
		1020	255	4:1	10			300	0.29	0.555 ms
		2040	255	8:1	10			450	0.22	1.03 ms

Comparison to HADES ([31]). Since the design of POSEIDON follows the same strategy as block ciphers in [31], we provide an explicit list of new material crafted for this paper:

- Hash-function specific (CICO, keyless, preimage) algebraic attacks, their analysis, and fixes against recent hash-only attacks
- Orientation towards various zero-knowledge proof systems and suggestions how to increase prover performance in these systems
- Instances for Merkle trees and variable-length hashing
- Concrete benchmarks for zero-knowledge proofs of accumulated values in Merkle trees, and a demonstration that it can be done in 1 second for billion-size trees

Related Work. The Zcash designers introduced a new 256-bit hash function called Pedersen hash [38, p.134], which is effectively a vectorized Pedersen commitment in elliptic curve groups with short vector elements. For the claimed 128-bit security level, it utilizes 869 constraints per 516-bit message chunk, thus having 1.7 constraints per bit, whereas our POSEIDON instances use from 0.2 to 0.45 constraints per bit, depending on the underlying prime field.

For the binary field case, Ashur and Dhooghe [8] have recently introduced the STARK-friendly block cipher JARVIS and its derivative hash function FRIDAY with several instances and security levels. They use a key-alternating structure with a single inverse S-box, followed by an affine transformation (with low degree in the extension field). However, both JARVIS and FRIDAY were successfully attacked shortly after their publication [3]. In the response, the authors created a new family of SNARK/STARK-friendly hash functions with *Vision* (binary fields) and *Rescue* (prime fields) being main instances [6]. The latter two share some similarity with our design with two important differences: First, all S-box layers are full (there are no partial rounds). Moreover, every second layer has S-boxes of the form $x^{1/d}$ for small d. This approach prevents some algebraic attacks but is also more expensive in software as the resulting power functions have high Hamming weight and thus require many squarings.

Structure of the Paper. We introduce POSEIDON as a HADES-based hash in Section 2 and follow up with real-world applications in Section 3. Concrete instances with round numbers and domain constants are given in Section 4. We summarize the cryptanalysis results in Section 5 and refer to [30, Appendix] for all the details. Finally, we estimate the performance of POSEIDON instances in zero-knowledge proof systems in Section 6 by computing R1CS (SNARK) and AET (STARK) costs.

Historic Remarks. We started working on the design of POSEIDON in the fall of 2018. The work was triggered by the STARK paper [9] where a Rijndael-based hash function was proposed for zero-knowledge applications, but we identified that the underlying cipher is not suitable for the hash mode due to related-key trails. In the design of POSEIDON, we were inspired by the LowMC cipher [5] with a partial S-box layer, the block cipher SHARK with its inverse S-box and its MDS matrix as the linear layer [50], and by MiMC with its

algebraically simple approach of using the cube S-box [4, 33]. We immediately considered a partial S-box layer for most of the rounds in order to gain performance and safe constraints. The S-box was initially either the inverse or a power map (as the cube function), but we later found out that the inverse function does not provide a sufficiently fast degree growth.

In 2019, we separated the design into two parts due to diverging analysis and use cases, namely the block cipher HADESMiMC and the hash functions POSEIDON and STARKAD. The latter was designed for binary fields, as we thought that they are useful for STARKs. However, it turned out that they are neither especially useful in this setting nor equally secure [14, 42], which is why we eventually dropped STARKAD.[4]

After the first publications of the design, we got requests from third parties to add explicit Merkle tree support and encryption (to be verifiable in zero knowledge). Later we were also asked to add weaker and stronger versions. Initially we allowed for greater flexibility in the choice of S-boxes, curves, width, etc., but only a few parameter sets are now given in the main body of this paper for the matter of user convenience: It turned out that too many possible parameters confuse users. Regarding zero-knowledge proof systems, we initially targeted Groth16 [35], Bulletproofs [19] and STARKs [9], and we later also added PLONK [27] due to its increased popularity.

2 The POSEIDON Hash Function

In the following, we propose the hash function POSEIDON, which maps strings over $\mathbb{F}_p$ (for a prime $p \approx 2^n$) to fixed-length strings over $\mathbb{F}_p$, i.e., POSEIDON : $\mathbb{F}_p^* \rightarrow \mathbb{F}_p^o$, where o is the output length measured in $\mathbb{F}_p$ elements (usually, $o = 1$). It is constructed by instantiating a sponge function with the POSEIDON$^\pi$ permutation. POSEIDON$^\pi$ is a variant of HADESMiMC proposed in [31], albeit instantiated with a fixed and known key.

We sometimes use the notation $p \approx 2^n$ and $N = n \cdot t \approx \log_2(p) \cdot t$ to denote the approximate size of the texts in bits.

2.1 Sponge Construction for POSEIDON$^\pi$

Sponges. A sponge construction [12] builds upon an internal permutation and can be used to achieve various goals such as encryption, authentication, or hashing. In addition to the internal permutation, it is usually defined by two parameters, namely the *rate* (or *arity* in the context of tree hashing) r and the capacity (or inner part) c. The rate determines the throughput, whereas the capacity is crucial for the security level. This means that, when fixing the size of the internal permutation to N bits, a tradeoff between throughput and security has to be made.

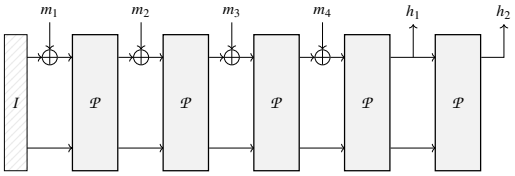

Figure 1: A sponge hash function.

An example for a sponge hash function is proposed in Fig. 1, where the construction is used to compute the hash output $h_1 \,||\, h_2$ of the 4-block message $m_1 \,||\, m_2 \,||\, m_3 \,||\, m_4$, where m_i and h_i are r-bit values. The initial state I contains all zeros, i.e., $I = 0^r \,||\, 0^c$ for an r-bit rate and a c-bit capacity.

Sponge Security. Depending on the properties of the N-bit internal permutation, a sponge construction allows to make strong arguments about the security of the overall design. Specifically, if this permutation is modeled as a randomly chosen permutation, the sponge function is indifferentiable from a random oracle for up to $2^{c/2}$ calls ($|\mathbb{F}|^{c/2}$ calls if the capacity is counted in field elements) [12]. A sponge hash function with a capacity of c bits can therefore provide $2^{c/2}$ bits of collision and $2^{c/2}$ bits of (second) preimage resistance.[5]

In this proposal, we instantiate the sponge function with our new permutation POSEIDON$^\pi$. Given the size N of the permutation and a desired security level s, we can hash $r = N - 2s$ bits per call to the permutation. Following this design strategy, we choose the number of rounds of the inner permutation POSEIDON$^\pi$ in order to ensure that such a permutation does not exhibit non-generic properties up to 2^M queries, where M is the desired security level.[6] For this we set the capacity to $2M$ and denote by POSEIDON-M a hash function that provides M bits of security against collision and preimage attacks.

Our POSEIDON$^\pi$ Sponges. We provide several POSEIDON instances for different use cases, but they all use the sponge construction in the same way as illustrated in Fig. 1:

1. Depending on the use case (Section 3), determine the capacity element value and the input padding if needed.
2. Split the input into chunks of size r.

[4] For reference, we recall STARKAD in [30, Appendix J].

[5] We present the Sponge construction over a binary field in order to follow the presentation made in [12]. It can easily be generalized for a prime field $\mathbb{F}_p^t$ by replacing each (N/t)-bit word by a $(\lceil log_2(p) \rceil)$-bit one.

[6] In other words, the permutation cannot be distinguished from a randomly drawn permutation.

3. Apply the permutation POSEIDON$^{\pi}$ to the capacity element and the first chunk.
4. Until no more chunks are left, add them into the state and apply the permutation.
5. Output o output elements out of the rate part of the state. If needed, iterate the permutation more times.

2.2 The HADES Design Strategy for Hashing

Cryptographic permutations usually consist of an efficient round function which is applied sufficiently many times in order to make the permutation behave like a randomly drawn one. In general, the same round function is used throughout the permutation, in order to destroy all of its possible symmetries and structural properties.

In HADES we consider different round functions within the same construction. More precisely, we mix rounds with *full S-box layers* and rounds with *partial S-box layers*. The motivation to have different types of rounds is that full S-box layers are expensive in software and ZK proof systems but are a good protection against statistical attacks, whereas partial layers are relatively cheap but are, in some cases, similarly good as full ones against algebraic attacks.

Details on the HADES Strategy. The HADES design strategy consists of R_f rounds in the beginning, in which S-boxes are applied to the full state. After these rounds, R_P rounds in the middle contain only a single S-box in each round, and the rest of the state goes through the non-linear layer unchanged (i.e., identity functions are used instead of the missing S-boxes). Finally, R_f rounds at the end are applied by again using S-boxes for the full state.

The idea of this approach is to provide arguments for the security against statistical attacks using the $R_F = 2R_f$ rounds with full S-box layers in the beginning and in the end together with the *wide trail strategy* [25], which is also used in, e.g., the AES [26]. On the other hand, the R_P rounds with partial S-box layers are a more efficient way to increase the degree of the overall function, and are mainly used for arguments against algebraic attacks.

A detailed overview of this approach is shown in Fig. 2.

The Round Function. Each round function of our POSEIDON permutation consists of the following three components.

1. *AddRoundConstants*, denoted by $ARC(\cdot)$
2. *SubWords*, denoted by S-box$(\cdot)$ or by $SB(\cdot)$
3. *MixLayer*, denoted by $M(\cdot)$

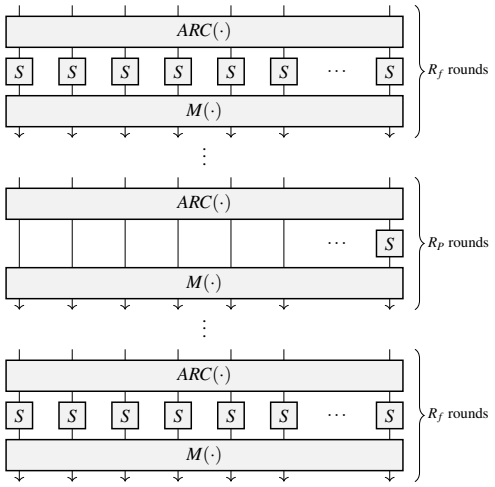

Figure 2: Construction of the HADES-based POSEIDON$^{\pi}$ permutation.

The *MixLayer* operation is the linear layer of our construction, and it consists in multiplying the state with a $t \times t$ MDS matrix in order to apply the wide trail strategy.

In total we get:

$$\underbrace{ARC \to SB \to M}_{\text{First round}} \to \cdots \to \underbrace{ARC \to SB \to M}_{(R-1)\text{-th round}} \to \underbrace{ARC \to SB \to M}_{R\text{-th round}}$$

While $ARC(\cdot)$ and $M(\cdot)$ are the same in each round, the number of S-boxes is not the same, namely

- $R_f + R_f = R_F$ rounds have *full* S-box layers, i.e., t S-box functions, and
- R_P rounds have *partial* S-box layers, i.e., 1 S-box and $(t-1)$ identity functions.

We refer to [31] for more details about the HADES design strategy.

Interaction Between Full and Partial Rounds. Note that the same number of full rounds can be used instead of the partial rounds without decreasing the security, but this leads to substantially higher costs in our target applications. However, replacing t partial rounds with one full round may keep the costs in our target applications similar, but the security may be severely decreased due to a significantly lower degree of 1 full round compared to t partial rounds.

2.3 The Permutation Family POSEIDON$^{\pi}$

The HADES design strategy provides a good starting point for our new hash function. Indeed, the combination of full and partial rounds allows us to make strong arguments about the security, while also exploiting the smaller number of S-boxes

in the partial rounds in order to gain efficiency in the target applications.

The primary application of our design is hashing in large prime fields, hence POSEIDON$^\pi$ takes inputs of $t \geq 2$ words in $\mathbb{F}_p$, where p is a prime of size $p \approx 2^n$ (i.e., $\lceil \log_2(p) \rceil = n$). We will now describe the components of each POSEIDON$^\pi$ round in detail.

The S-Box Layer. For the applications we have in mind, we focus on two S-boxes.

- First, we consider the α-power S-box, defined by S-box$(x) = x^\alpha$, where α is the smallest positive integer s.t. $\gcd(\alpha, p-1) = 1$. In the following, these permutations are called "x^α-POSEIDON$^\pi$". Examples are given by $\alpha = 3$ (x^3-POSEIDON$^\pi$) if $p \neq 1 \mod 3$ or $\alpha = 5$ (x^5-POSEIDON$^\pi$) if $p \neq 1 \mod 5$.
- Secondly, we consider the inverse S-box$(x) = x^{-1}$ (under the assumption S-box$(0) = 0$). In the following, these permutations are called "x^{-1}-POSEIDON$^\pi$".

It turns out that the S-box x^5 is suitable for two of the most popular prime fields in ZK applications, concretely the prime subfields of the scalar field of the BLS12-381 and BN254 curves, so we mainly consider this S-box, but try to present generic cryptanalytic results for other cases whenever possible.

The Linear Layer. A $t \times t$ MDS matrix[7] with elements in $\mathbb{F}_p$ exists if the condition (see [45] for details)

$$2t + 1 \leq p$$

is satisfied.

Given p and t, there are several ways to construct an MDS matrix. One of them is using a Cauchy matrix [53], which we recall here briefly. For $x_i, y_i \in \mathbb{F}_p$, where $i \in [1,t]$, the entries of the matrix $\mathcal{M}$ are defined by

$$\mathcal{M}_{i,j} = \frac{1}{x_i + y_j},$$

where the entries of $\{x_i\}_{1 \leq i \leq t}$ and $\{y_i\}_{1 \leq i \leq t}$ are pairwise distinct and $x_i + y_j \neq 0$, where $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, t\}$.

Avoiding Insecure Matrices. We emphasize that not every MDS matrix provides the same level of security. In particular, the matrix $\mathcal{M}$ must prevent the possibility to set up

(1) invariant (or iterative) subspace trails [32] (or equivalently, truncated differentials) with prob. 1 with inactive S-boxes over more than $t-1$ rounds[8] (more details are given in the following), or

(2) invariant (or iterative) subspace trails with prob. 1 and with active S-boxes for any number of rounds.

Regarding the first point, let $\mathcal{S}^i$ be the subspace s.t. no S-box is active in the first i consecutive rounds, that is,

$$\mathcal{S}^{(i)} := \left\{ v \in \mathbb{F}^t \;\middle|\; [\mathcal{M}^j \cdot v]_0 = 0 \in \mathbb{F}, j < i \right\}, \qquad (1)$$

where $[x]_0$ denotes the first word of $x \in \mathbb{F}^t$, $\mathcal{S}^{(0)} = \mathbb{F}^t$, and $\dim\left(\mathcal{S}^{(i)}\right) \geq t - i$. For each pair of texts (x, y) in the same coset of $\mathcal{S}^i$, no S-boxes are active in the first i consecutive rounds. Hence, a truncated differential with prob. 1 (or equivalently, a subspace trail) can be set up for the first $i \leq t-1$ rounds. The matrix $\mathcal{M}$ must be chosen s.t. no subspace trail with inactive/active S-boxes can be set up for more than $t-1$ rounds.

A detailed analysis of matrix properties related to this attack vector can be found in [34]. With these results in mind, we suggest the following method to generate matrices:

1. Randomly select pairwise distinct $\{x_i\}_{1 \leq i \leq t}$ and $\{y_i\}_{1 \leq i \leq t}$, where $x_i + y_j \neq 0$ and where $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, t\}$.
2. Determine if the matrix is secure using Algorithm 1, Algorithm 2, and Algorithm 3 provided[9] in [34]. For a secure matrix, no infinitely long (invariant and/or iterative) subspace trail (with or without active S-boxes) can be set up for all rounds with partial S-box layers.
3. Repeat this procedure until a secure matrix is found.

We used this method to generate the matrices for the instantiations given in Section 4. For [34, Algorithm 3], we used a search period of $l = 4t$, and we additionally made sure that no invariant subspace trails with active S-boxes exist for $M, M^2, \ldots, M^l$. In our experiments, we observed that only a few trials are needed in order to find a secure matrix for sufficiently large fields.

To summarize, this approach allows us to make sure that our MDS matrices do not exhibit the vulnerabilities discussed in [34], and our instantiations are thus secure against this specific type of attack.

Efficient Implementation. We refer to [30, Appendix B] for details about efficient POSEIDON$^\pi$ implementations. The

[7] A matrix $M \in \mathbb{F}^{t \times t}$ is called *maximum distance separable* (MDS) iff it has a branch number $\mathcal{B}(M)$ equal to $\mathcal{B}(M) = t + 1$. The branch number of M is defined as $\mathcal{B}(M) = \min_{x \in \mathbb{F}^t} \{\mathrm{wt}(x) + \mathrm{wt}(M(x))\}$, where wt is the Hamming weight in wide trail terminology. Equivalently, a matrix M is MDS iff every submatrix of M is non-singular.

[8] This fixes a weakness in the previous version of POSEIDON, where specific choices of $\mathcal{M}$ over $(\mathbb{F}_p)^t$ could have resulted in vulnerable instances. We refer to [14, 42] for more details.

[9] https://extgit.iaik.tugraz.at/krypto/linear-layer-tool

main advantage of these strategies consists of reducing the number of constant multiplications in each round with a partial S-box layer from t^2 to $2t$, which is particularly useful for large t and R_P. For example, we implemented x^3-POSEIDON$^\pi$ with $(n, t, R_F, R_P) = (64, 24, 8, 42)$ in `Sage`, and we could observe that the performance improves by a factor of about 5, with the average computation time being 4 ms for the optimized version.

3 Applications

We suggest POSEIDON for all applications of zero-knowledge-friendly hashing, concretely:

- Using POSEIDON for commitments in various protocols, where the knowledge of the committed value is proven in zero knowledge: For this we suggest a single-call permutation-based hashing with POSEIDON-128 and widths from 2 to 5 field elements. The advantage over the Pedersen hash, for example, is that POSEIDON is faster and can also be used in signature schemes which allows for a smaller code footprint.
- Hashing multi-element objects with certain fields encoded as field elements, so that statements about these fields are proven in zero knowledge: We suggest variable-length sponge-based hashing with POSEIDON-128 or POSEIDON-80 with width 5 (and rate 4).
- Using POSEIDON in Merkle trees to enable zero-knowledge proofs of knowledge of a leaf in the tree with optional statements about the leaf content: We recommend Merkle trees of arity 4 (i.e., width 5) with POSEIDON-128 as the most performant, but trees of more conventional arities can be used as well.
- Verifiable encryption with POSEIDON within Integrated Encryption Scheme [28]: Put POSEIDON inside the DuplexSponge authenticated encryption framework [13] and initialize it with a session key based on the recipient's public key. Then one can prove that the recipient can decrypt the ciphertext into a plaintext with certain properties.

There exist several third-party protocols that already use POSEIDON in these use cases:

- Filecoin employs POSEIDON for Merkle tree proofs with different arities and for two-value commitments.[10]
- Dusk Network uses POSEIDON to build a Zcash-like protocol for securities trading.[11] It also uses POSEIDON for encryption as described above.
- Sovrin uses POSEIDON for Merkle-tree based revocation [2].
- Loopring uses POSEIDON for private trading on Ethereum.[12]

4 Concrete Instantiations of POSEIDON$^\pi$

As of mid-2020, many protocols that employ zero-knowledge proofs use (or plan to use) pairing-based proof systems [23, 27, 35, 49] or Bulletproofs [19]. The elliptic curves used by these systems are predominantly BLS12-381, BN254, and Ed25519. A hash function friendly for such a system would operate in the scalar prime field of the curve, and they all have a size of around 2^{255}.

4.1 Main Instances

We present POSEIDON$^\pi$ permutations for such prime fields, and leave the other cases to [30, Appendix]. The S-box function can be chosen as x^5 in all cases, i.e., we use x^5-POSEIDON for hashing in all such protocols, though the concrete fields are slightly different (this affects only constants and matrices, but not the number of rounds).

The security levels M of 80 and 128 bits correspond to a 255-bit capacity, i.e., one field element. We focus on two possible widths, namely $t = 3$ and $t = 5$, as they correspond to popular cases of 2-to-1 and 4-to-1 compression functions. In the Merkle tree case, this corresponds to trees of arity 2 and 4, respectively. The round numbers for 80- and 128-bit security levels are given in Table 2, and a more extensive set is given in [30, Appendix G]. For $M = 256$ we select a capacity and an output of 2 255-bit elements (one 510-bit element is possible too).

All our MDS matrices are Cauchy matrices, and the method to construct them is further described in Section 2.3. We use sequences of integers for the construction.

The round constants and matrices are generated using the Grain LFSR [37] in a self-shrinking mode, and the detailed initialization and generation are described in [30, Appendix F]. Using this method, the generation of round constants and matrices depends on the specific instance, and thus different round constants are used even if some of the chosen parameters (e.g., n and t) are the same. Note that by using the Grain LFSR and instance-specific seed values, this approach is reminiscent of nothing-up-my-sleeve (NUMS) numbers. Indeed, letting the attacker freely choose round constants and/or matrices can lead to attacks.

[10] https://github.com/filecoin-project/neptune
[11] https://github.com/dusk-network/Poseidon252
[12] https://tinyurl.com/y7tl537o

Table 2: Concrete instantiations of POSEIDON$^{\pi}$ (with security margin) over BLS12-381, BN254, Ed25519 scalar fields.

Instance (S-box: $f(x) = x^5$)	t	R_F	R_P
POSEIDON$^{\pi}$-128	3	8	57
	5	8	60
POSEIDON$^{\pi}$-80	3	8	33
	5	8	35
POSEIDON$^{\pi}$-256	6	8	120
	10	8	120

We provide the round constants, the matrices, and test vectors in auxiliary files for four primary instantiations. We also make reference implementations for these instantiations and scripts to calculate the round numbers, the round constants, and the MDS matrices available online.[13] We refer to [30, Appendix A] for a more detailed overview of the auxiliary files.

4.2 Domain Separation for POSEIDON

POSEIDON can be used in a number of applications, and having the same instance for all of them is suboptimal. Also, some protocols explicitly require several different hash functions. We suggest using domain separation for this, concretely encoding the use case in the capacity element (which is fine as it is 256 bits large and has a lot of bits to fill) and using some padding to distiguish inputs of different lengths if they may happen. Although a concrete form of domain separation constants is not security critical, we suggest a common methodology to unify potential implementations.

Concretely, we propose:

- **Merkle Tree** (all leafs are present, up to arity 32). The capacity is $2^{arity} - 1$. A generic case is considered in [30, Appendix I]. We use no padding here.
- **Merkle Tree** (some leafs may be empty). The capacity value equals the bitmask of which leafs are present. We use no padding here.
- **Variable-Input-Length Hashing**. The capacity value is $2^{64} + (o-1)$ where o the output length. The padding consists of one field element being 1, and the remaining elements being 0.
- **Constant-Input-Length Hashing**. The capacity value is $length \cdot (2^{64}) + (o-1)$ where o the output length. The padding consists of the field elements being 0.
- **Encryption**. The capacity value is 2^{32}. The padding consists of the field elements being 0.
- **Future Uses**. The capacity value is $identifier \cdot (2^{32})$. The padding depends on the application.

[13] https://extgit.iaik.tugraz.at/krypto/hadeshash

5 Cryptanalysis Summary of POSEIDON

As for any new design, it is paramount to present a concrete security analysis. In the following, we provide an in-depth analysis of the security of our construction. Due to a lack of any method to ensure that a hash function based on a sponge construction is secure against all possible attacks, we base our argumentation on the following consideration. As we just recalled in the previous section, when the internal permutation $\mathcal{P}$ of an $(N = c + r)$-bit sponge function is modeled as a randomly chosen permutation, the sponge hash function is indifferentiable from a random oracle up to $2^{c/2}$ calls to $\mathcal{P}$. Thus, we choose the number of rounds of the inner permutation case in order to provide security against distinguishers relevant to collision/preimage attacks. Equivalently, this means that such a number of rounds guarantees that $\mathcal{P}$ does not exhibit any relevant non-random/structural properties (among the ones known in the literature).

5.1 Definitions

Definition 5.1. The function F is T-secure against collisions if there is no algorithm with expected complexity smaller than T that finds x_1, x_2 such that $F(x_1) = F(x_2)$.
Definition 5.2. The function F is T-secure against preimages if there is no algorithm with expected complexity smaller than T that for given y finds x such that $F(x) = y$.
Definition 5.3. The function F is T-secure against second preimages if there is no algorithm with expected complexity smaller than T that for given x_1 finds x_2 such that $F(x_1) = F(x_2)$.
Definition 5.4. The invertible function P is T-secure against the CICO (m_1, m_2)-problem if there is no algorithm with expected complexity smaller than T that for given m_1-bit I_1 and m_2-bit O_1 finds I_2, O_2 such that $P(I_1||I_2) = P(O_1||O_2)$.

5.2 Security Claims

In terms of concrete security, we claim that POSEIDON-M is 2^M-secure against collisions and (second) preimages. To help increase confidence in our design and simplify external cryptanalysis, we also explicitly state another claim about our internal permutation: POSEIDON$^{\pi}$ is $2^{\min(M, m_1, m_2)}$-secure against the CICO (m_1, m_2)-problem.

Even though an attack below these thresholds may not affect any concrete applications of our hash functions, we would still consider it an important cryptanalytic result.

5.3 Summary of Attacks

Here we list the main points of our cryptanalysis results. The number of rounds $R = R_P + R_F$ we can break depends on the security level M and the number of S-boxes t, which we specify for each concrete hash function instance in the next section.

Before going on, we point out that for all attacks that are in common to the ones proposed for the cipher HadesMiMC [31], here we limit ourselves to report the main idea and result. For all other cases (namely, higher-order differentials, zero-sum partitions, Gröbner basis attacks, and preimage attacks), we present here more details. In any case, all details are provided in [30, Appendix].

We highlight that the following cryptanalysis is not equivalent to the one presented for the block cipher HADESMiMC. Indeed, the scenarios are different (in one case the goal is to guarantee the impossibility to find the secret key, while here there is no secret key material and the goal is to guarantee that the internal permutation looks like a pseudo-random permutation). This means that certain attacks that we consider here are not valid in the case of a block cipher and vice-versa. Just to give some examples, the rebound attack [44, 48] holds only in the context studied here, while a MitM scenario (crucial in the case of an SPN cipher) does not work in the context of a sponge function, since the attacker does not know the full output. More details are given in the following.

Proposition 5.1 (Informal)**.** The following number of rounds for x^5-POSEIDON-128 over $\mathbb{F}_p$ with $\approx$ 256-bit p protects against statistical and algebraic attacks:

$$R_F = 6, \qquad R = R_F + R_P = 56 + \lceil \log_5(t) \rceil.$$

Proof. We substitute $\alpha = 5, M = 128$ and $\log_2(p) = 255$ to Equations (2),(3),(5) and see that no one is satisfied, i.e., the attacks do not work. □

Proposition 5.2 (Informal)**.** The following number of rounds for x^5-POSEIDON-80 over $\mathbb{F}_p$ with $\approx$ 256-bit p protects against statistical and algebraic attacks:

$$R_F = 6, \qquad R = R_F + R_P = 35 + \lceil \log_5(t) \rceil.$$

Proposition 5.3 (Informal)**.** The following number of rounds for x^5-POSEIDON-256 over $\mathbb{F}_p$ with $\approx$ 256-bit p protects against statistical and algebraic attacks:

$$R_F = 6, \qquad R = R_F + R_P = 111 + \lceil \log_5(t) \rceil.$$

5.4 Security Margin

Given the *minimum* number of rounds necessary to provide security against all attacks known in the literature, we *arbitrarily* decided to add

(1) two more rounds with full S-box layers, and

(2) 7.5% more rounds with partial S-box layers,

i.e., $+2\ R_F$ and $+7.5\%\ R_P$. The resulting number of rounds for our primary instances is given in Table 2.

5.5 Attack details

All the attacks below are applied to the internal permutation POSEIDON$^\pi$. The sponge framework dictates that all the attacks on the hash function with complexity below $2^{c/2}$ must result from attacks on the permutation. Thus we show that no such attack on the permutation should exist.

5.5.1 Statistical Attacks

Differential/Linear Distinguishers. As shown in the appendix, at least 6 rounds with full S-box layers are necessary to provide security against the statistical attacks we consider. In more detail, for

$$R_F < \begin{cases} 6 & \text{if } M \leq (\lfloor \log_2 p \rfloor - \mathcal{C}) \cdot (t+1) \\ 10 & \text{otherwise} \end{cases} \tag{2}$$

linear [47] and differential [16, 17] attacks may be possible, where $\mathcal{C} = 2$ for $S(x) = 1/x$ and $\mathcal{C} = \log_2(\alpha - 1)$ for $S(x) = x^\alpha$ (where remember that α is an odd integer number), e.g., $\mathcal{C} = 1$ for $S(x) = x^3$ and $\mathcal{C} = 2$ for $S(x) = x^5$.

Before going on, we highlight that we exploit only rounds with full S-box layers in order to prevent statistical attacks (as done in [31]). As explained in [42], under the assumption made for the linear layer in Section 2.3, it is possible to exploit both the rounds with partial and full S-box layers in order to guarantee security against some statistical attacks, like differential and linear attacks. Our decision to consider only rounds with full S-box layers has been made since a similar condition on the rounds with full S-box layers (e.g., $R_F \geq 6$) is necessary for the security against some algebraic attacks (e.g., Gröbner basis attacks – see in the following) and in order to provide simple security arguments for all statistical attacks (including e.g. the rebound one).

(Invariant) Subspace Trails. We emphasize that the choice of the matrix that defines the linear layer, made in Section 2.3, prevents the existence of subspaces $\mathcal{S}$ that generate infinitely long subspace trails, namely a finite collection of subspaces $\{\mathcal{S}_0, \ldots, \mathcal{S}_{r-1}\}$ s.t. each coset of $\mathcal{S}_i$ is mapped into a coset of $\mathcal{S}_{i+1}$ with probability 1 (where the index is taken modulo r) an arbitrary number of times. This allows to fix the weakness of the previous version of POSEIDON.

Other Attacks. Finally, we briefly mention that the same number of rounds given before for the case of differential/linear attacks guarantees security against other attacks as truncated differentials [43], impossible differentials [15], rebound attacks [44, 48], and so on. More details are given in [30, Appendix].

5.5.2 Algebraic Attacks

In order to estimate the security against algebraic attacks, we evaluate the degree of the reduced-round permutations and their inverses. Roughly speaking, our results can be summarized as follows, where $n \simeq \log_2(p)$.

Interpolation Attack. The interpolation attack [39] depends on the number of different monomials in the interpolation polynomial, where (an upper/lower bound of) the number of different monomials can be estimated given the degree of the function. The idea of such an attack is to construct an interpolation polynomial that describes the function. If the number of unknown monomials is sufficiently large, then this cannot be done faster than via a brute-force attack.

For a security level of M bits, the number of rounds that can be attacked is

- for $S(x) = x^\alpha$:

$$R \leq \lceil \log_\alpha(2) \cdot \min\{M, \log_2(p)\} \rceil + \lceil \log_\alpha t \rceil \quad (3)$$

- for $S(x) = 1/x$:

$$\lfloor R_F \log_2(t) \rfloor + R_P \leq \lceil \log_2(t) \rceil + \lceil 0.5 \cdot \min\{M, \log_2(p)\} \rceil \quad (4)$$

In general, the number of unknown monomials does not decrease when increasing the number of rounds. Hence, a higher number of rounds likely leads to a higher (or equal) security against this attack. We also consider various approaches of the attack (such as the MitM one) in [30, Appendix C.2.1].

Gröbner Basis Attack. In a Gröbner basis attack [24], one tries to solve a system of non-linear equations that describe the function. The cost of such an attack depends on the degree of the equations, but also on the number of equations and on the number of variables. Since there are several ways for describing the studied permutation, there are several ways to set up such a system of equations and so the attack. Here, we focus on two extreme cases:

1. In the first case, the attacker derives equations, one for each word, for the entire r-round permutation. Assuming $S(x) = x^\alpha$ (analogous for the others), we show that the attack complexity is about α^{2t} (see below), therefore for a security level of M bits the attack works at most on $\log_\alpha 2^{\min\{n/2, M/2\}}$ rounds.
2. In the second case, since a partial S-box layer is used, it may be more efficient to consider degree-α equations for single S-boxes. In this case, more rounds can be necessary to guarantee security against this attack.

In both cases, it is possible to make use of the existence of the subspace $\mathcal{S}^{(r)}$ defined as in Eq. (1) in order to improve the attack. As shown in [14], such a subspace can be exploited in order to replace some non-linear equations of the system that we are trying to solve with linear equations. Indeed, given a text in a coset of the subspace $\mathcal{S}^{(r)}$, the output of such a text after r rounds with partial S-box layers is simply the result of an affine map applied to the input (i.e., no S-box is involved). As explained in detail in [30, Appendix C.2.2], this issue can easily be fixed both by a careful choice of the matrix that defines the linear layer (see Section 2.3 for details) and, if necessary, by adjusting the number of rounds with partial S-box layers.

With optimistic (for the adversary) complexity of the Gaussian elimination, we obtain for each S-box two attacks which are faster than 2^M if either condition is satisfied:

- if $S(x) = x^\alpha$:

$$\begin{cases} R \leq \log_\alpha(2) \cdot \min\left\{\frac{M}{3}, \frac{\log_2(p)}{2}\right\}, \\ R \leq t - 1 + \min\left\{\frac{\log_\alpha(2) \cdot M}{t+1}, \frac{\log_\alpha(2) \cdot \log_2(p)}{2}\right\} \end{cases} \quad (5)$$

- if $S(x) = 1/x$:

$$\begin{cases} \lfloor R_F \log_2(t) \rfloor + R_P \leq \lceil 0.5 \cdot \min\{M, \log_2(p)\} \rceil + \lceil \log_2(t) \rceil \\ \lfloor R_F \log_2(t) \rfloor + R_P \leq t - 1 + \lceil \log_2(t) \rceil + \\ \qquad + \min\left\{\left\lceil \frac{M}{t+1} \right\rceil, \lceil 0.5 \cdot \log_2(p) \rceil\right\} \end{cases} \quad (6)$$

Higher-Order Differential Attack. Working over $\mathbb{F}_{2^n}{}^t \equiv \mathbb{F}_2^{n \cdot t}$, the higher-order differential attack [43] depends on the *algebraic degree* of the polynomial function that defines the permutation, where the algebraic degree δ of a function $f(x) = x^d$ of degree d over $\mathbb{F}_{2^n}$ is defined as $\delta = \mathrm{hw}(d)$ (where $\mathrm{hw}(\cdot)$ is the Hamming weight). The idea of such an attack is based on the property that given a function $f(\cdot)$ of algebraic degree δ, $\bigoplus_{x \in \mathcal{V} \oplus \phi} f(x) = 0$ if the dimension of the subspace $\mathcal{V}$ satisfies $\dim(\mathcal{V}) \geq \delta + 1$. If the algebraic degree is sufficiently high, the attack does not work.

At first thought, one may think that this attack does not apply (or is much less powerful) in $\mathbb{F}_p^t$ (due to the fact that the only subspaces of $\mathbb{F}_p$ are $\{0\}$ and $\mathbb{F}_p$ itself). Recently, it has been shown in [14] how to set up an higher-order differential over $\mathbb{F}_p^t$. Given f over $\mathbb{F}_p$ of degree $d \leq p - 2$, $\sum_{x \in \mathbb{F}_p} f(x) = 0$.

Since this result is related to the degree of the polynomial that describes the permutation, we claim that the number of rounds necessary to provide security against the interpolation attack provides security against this attack as well.

(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 [18] for two SHA-3 candidates, *Luffa* and KECCAK. More generally, a zero-sum structure for a function $f(\cdot)$ is defined as a set Z of inputs z_i that sum to zero, and for which the corresponding outputs $f(z_i)$ also sum to zero, i.e., $\sum_i z_i = \sum_i f(z_i) = 0$. 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 5.5 (Zero-Sum Partition [18])**.** Let P be a permutation over $\mathbb{F}_q^t$ for a prime $q \geq 2$. A zero-sum partition for P of size $K < t$ is a collection of K disjoint sets $\{X_1, \ldots, X_K\}$ with the following properties:

- $X_i \subset \mathbb{F}^t$ for each $i = 1, \ldots, k$ and $\bigcup_{i=1}^{k} X_i = \mathbb{F}^t$,
- $\forall i = 1, \ldots, K$: the set X_i satisfies the zero-sum property $\sum_{x \in X_i} x = \sum_{x \in X_i} P(x) = 0$.

Here we explicitly state that we do not make claims about the security of POSEIDON$^\pi$ against zero-sum partitions. This choice is motivated by 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. As a concrete example, consider the case of KECCAK: While 24 rounds of KECCAK-f can be distinguished from a random permutation using a zero-sum partition [18] (that is, *full* KECCAK-f), preimage/collision attacks on KECCAK can only be set up for up to 6 rounds of KECCAK-f [36]. This hints that zero-sum partitions should be largely ignored for practical applications.

For completeness, we mention that a zero-sum partition on (a previous version of) reduced-round POSEIDON$^\pi$ has been proposed in [14]. Such a property can cover up to $R_F = 6$ rounds (i.e., 2 rounds at the beginning and 4 rounds at the end) by exploiting the inside-out approach and by choosing a subspace of texts after the first R_f rounds with full S-box layers and before the R_P rounds with partial S-box layers. Since the number of rounds of this new version is not smaller than the number of rounds of the previous one, and since $R_F \geq 8$ (see Section 5.4), it seems that a zero-sum partition cannot be set up for full POSEIDON$^\pi$.

6 POSEIDON in Zero-Knowledge Proof Systems

Our hash functions have been designed to be friendly to zero-knowledge applications. Specifically, we aim to minimize the proof generation time, the proof size, and the verification time (when it varies). Before presenting concrete results, we give a small overview of ZK proof systems to date.

6.1 State of the Art

Let $\mathcal{P}$ be a circuit over some finite field $\mathbb{F}$ where gates are some (low-degree) polynomials over $\mathbb{F}$ with I and O being input and output variables, respectively: $\mathcal{P}(I) = O$. The *computational integrity problem* consists of proving that some given O_0 is the result of the execution of $\mathcal{P}$ over some I_0: $\mathcal{P}(I_0) = O_0$. It is not difficult to show that any limited-time program on a modern CPU can be converted to such a circuit [10], and making the proof zero-knowledge is often possible with little overhead.

The seminal PCP series of papers states that for any program $\mathcal{P}$ it is possible to construct a proof of computational integrity, which can be verified in time sublinear in the size of $\mathcal{P}$. However, for a long time the prover algorithms were so inefficient that this result remained merely theoretical. Only recently, proof systems where the prover costs are polynomial in $|\mathcal{P}|$ were constructed, but they required a trusted setup: a verifier or someone else (not the prover) must process the circuit with some secret s and output a reference string S, used both by the prover and the verifier. In this setting, the prover's work can even be made linear in $|\mathcal{P}|$, and the verifier's costs are constant. These systems were called SNARKs for proof succinctness. The first generation of SNARKs, known as Pinocchio and Groth16 [35, 49], require a separate trusted setup for each circuit. The next generation, which includes Sonic [46], PLONK [27], and Marlin [23], can use one reference string of size d for all circuits with at most d gates, thus simplifying the setup and its reuse. Later on, proof systems without trusted setups appeared, of which we consider Bulletproofs [19], STARKs [9], and RedShift [41] the most interesting, though all of them come with deficiencies: Bulletproofs have linear verifier times (but rather short proofs), STARKs work with iterative programs, and RedShift has large proofs (up to 1 MB for millions of gates).

Current benchmarks demonstrate that programs with millions of gates can be processed within a few seconds with the fastest proof systems, which solves the computational integrity problem for some practical programs. Among them, privacy-preserving cryptocurrencies, mixers, and private voting are prominent examples. In short, such applications work as follows:

1. Various users add publicly hashes of some secret and public values to some set V, which is implemented as a Merkle tree. Hashes can be currency transaction digests, public keys, or other credentials.
2. Only those who know a secret behind some hash are declared eligible for an action (e.g., to vote or to spend money).
3. A user who wants to perform the action proves that they know a tree leaf L and a secret K such that L is both the hash of K and a leaf in V. If the proof passes, the user is allowed to perform an action (e.g., to vote). If an action must be done only once, a deterministic hash of the secret and leaf position can be computed and published.

This paradigm is behind the cryptocurrency Zcash and Ethereum mixers.

The bottleneck of such a system is usually the proof creation time, which took 42 seconds in the early version of Zcash, and sometimes the verifier's time. Both are determined by the size of the circuit that describes a Merkle proof and are thus dependent on the hash function that constitutes the tree.

Unfortunately, a single hash function cannot be optimal for all ZK proof systems, because they use different arithmetizations: STARKs can use prime and binary fields, Bulletproofs uses any prime field, whereas most SNARKs use a prime field based on a scalar field of a pairing-friendly elliptic curve. Therefore, for each proof system a new instance of POSEIDON may be needed. In the following we describe how this is done and how to optimize a circuit for some proof systems.

6.2 SNARKs with POSEIDON$^\pi$

In SNARKs, the prime field is typically the scalar field of some pairing-friendly elliptic curve. The primitive POSEIDON$^\pi$ can be represented as such a circuit with reasonably few gates, but the parameters of POSEIDON$^\pi$ must have been determined first by p. Concretely, after p is fixed, we first check if x^α is invertible in GF(p), which is true if $p \bmod \alpha \neq 1$. If this inequality is not satisfied for a small α, we use the inverse S-box or consider another prime power for the S-box.

6.2.1 Groth16

Groth16 [35] is an optimization of the Pinocchio proof system and currently the fastest SNARK with the smallest proofs. The Groth16 prover complexity is $O(s)$, where s is the number of rank-1 constraints – quadratic equations of the form $(\sum_i u_i X_i)(\sum_i v_i X_i) = \sum_i w_i X_i$, where u_i, v_i, w_i are field elements and X_i are program variables. It is easy to see that the S-box x^3 is represented by 2 constraints, the S-box x^5 by 3 constraints, and the S-box $1/x$ by 3 constraints (1 for the non-zero case, and two more for the zero case). Thus, in total we have

$$2tR_F + 2R_P \text{ constraints for } x^3\text{-POSEIDON}^\pi,$$
$$3tR_F + 3R_P \text{ constraints for } x^5\text{-POSEIDON}^\pi,$$
$$3tR_F + 3R_P \text{ constraints for } x^{-1}\text{-POSEIDON}^\pi.$$

It requires a bit more effort to see that we do not need more constraints as the linear layers and round constants can be incorporated into these ones. However, it is necessary to do some preprocessing. For example, in the POSEIDON$^\pi$ setting, the full S-box layers are followed by a linear transformation M. Each round with a full S-box layer can be represented by the following constraints in the SNARK setting:

$$\left(\sum_j M_{i,j} x_{i,j}\right) \cdot \left(\sum_j M_{i,j} x_{i,j}\right) = y_i \quad 1 \le i \le t,$$
$$y_i \cdot \left(\sum_j M_{i,j} z_{i,j}\right) = z_i,$$

where $M = I_{t \times t}$ for the first round. However, in a round with a partial S-box layer, we will have only one such constraint for $j = 1$. For the rest of the $t-1$ variables we will have linear constraints of the form

$$\sum_j M_{i,j} x_{i,j} = u_i \quad \text{, where } \ 2 \le i \le t.$$

Since the linear constraints have little complexity effect in Groth16, in the partial S-box rounds they can be composed with the ones from the previous round(s) using

$$\sum_k M_{i,k} \left(\sum_j M_{i,j} x_{i,j}\right) = v_k \quad 2 \le k \le t.$$

We can now calculate the number of constraints for the sponge mode of operation and for Merkle trees. In sponges, the $2M$ bits are reserved for the capacity, so $N - 2M$ bits are fed with the message. Therefore, we get

- $\frac{2tR_F + 2R_P}{N-2M}$ constraints per bit for x^3-POSEIDON$^\pi$,
- $\frac{3tR_F + 3R_P}{N-2M}$ constraints per bit for x^5-POSEIDON$^\pi$,
- $\frac{3tR_F + 3R_P}{N-2M}$ constraints per bit for x^{-1}-POSEIDON$^\pi$.

For the Merkle tree, we suggest a 1-call sponge where all branches must fit into the rate. Then a Merkle tree has arity $\frac{N}{2M} - 1$. Based on that we can calculate how many constraints we need to prove the opening in a Merkle tree of, for example, 2^{32} elements (the recent ZCash setting). The tree will have $\frac{32}{\log_2[N/(2M)-1]}$ levels with the number of constraints in each according to the above. The `libsnark` performance of the POSEIDON preimage prover (proof that for given y you know x such that $H(x) = y$) is given in Table 3. These experiments

Table 3: `libsnark` [1] performance of the POSEIDON preimage prover (one permutation call). Here t denotes the width.

Field	Arity (t)	`libsnark` ZK proof time for one hash		R1CS constraints
		Prove	Verify	
POSEIDON-128				
BN254	2:1 (3)	43.1ms	1.2ms	276
	4:1 (5)	57.9ms	1.1ms	440
POSEIDON-80				
BN254	2:1 (3)	32.8ms	1.2ms	180
	4:1 (5)	46.9ms	1.1ms	290

were performed on a desktop with an Intel Core i7-8700 CPU (@3.2GHz) and 32 GiB of memory.

As an example, we calculate the concrete number of constraints for a Merkle tree proof, where the tree has 2^{30} elements, assuming a security level of 128 bits and a prime field of size close to 2^{256}. We take the S-box equal to x^5 as it fits many prime fields: Ristretto (the prime group based on the scalar field of Ed25519), BN254, and BLS12-381 scalar fields. The results are in Table 4.

6.2.2 Bulletproofs

Bulletproofs [19] is a proof system that does not require a trusted setup. It is notable for short proofs which are logarithmic in the program size, and also for the shortest range proofs that do not require a trusted setup. However, its verifier is linear in the program size. For the use cases where the trusted setup is not an option, the Bulletproofs library "dalek" is among the most popular ZK primitives. We have implemented[14] a Merkle tree prover for POSEIDON in Bulletproofs using the same constraint system as for Groth16 with results outlined in Table 5. The performance varies since the underlying curves are based on prime fields of different size and weight: BN254 uses a 254-bit prime whereas BLS12-381 uses a 381-bit one (the reason for that is the recent reevaluation of discrete logarithm algorithms specific to pairing-friendly curves).

6.2.3 PLONK

PLONK [27] is a novel but popular SNARK using a universal trusted setup, where a structured reference string of size d can be used for any circuit of d gates or less. The setup is pretty simple as for the secret k the values $\{k^i \cdot B\}_{i \leq d}$ are stored, where B is an elliptic curve point and $\cdot$ denotes scalar multiplication. A PLONK proof is a combination of KZG polynomial commitments [40] and their openings, both using the SRS.

[14] https://github.com/lovesh/bulletproofs-r1cs-gadgets/blob/master/src/gadget_poseidon.rs

Table 4: Number of R1CS constraints for a circuit proving a leaf knowledge in the Merkle tree of 2^{30} elements.

Arity	Width	R_F	R_P	Total constraints
POSEIDON-128				
2:1	3	8	57	7290
4:1	5	8	60	4500
8:1	9	8	63	4050
Rescue-x^5				
2:1	3	16	-	8640
4:1	5	10	-	4500
8:1	9	10	-	5400
Pedersen hash				
510	171	-	-	41400
SHA-256				
510	171	-	-	826020
Blake2s				
510	171	-	-	630180
MiMC-$2p/p$ (Feistel)				
1:1	2	324	-	19440

The standard version of PLONK works with the same constraint system as we have described, plus it uses special machinery to lay out wires in the circuit. A prover first crafts three polynomials of degree equal to the number of gates, which are responsible for the left input, the right input, and the output, respectively. Then they allocate several supplementary polynomials to describe the wire layout. The prover complexity for a POSEIDON$^\pi$ permutation with the S-box x^5 of width w and R rounds is $11(w(w+6)+3)R$ point multiplications, and the proof has 7 group elements and 7 field elements. A third-party non-optimized implementation of a PLONK prover in Rust (Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz) gives us benchmarks, which we provide in Table 6.

As we have almost identical rounds, the PLONK compiler can be heavily optimized. Concretely, we suggest the following.

- Define a separate polynomial for each S-box line.
- Get rid of wire layout polynomials.
- Express round transitions as a system of affine equations

Table 5: Bulletproofs performance to prove 1 out of 2^{30}-Merkle tree.

Field	Arity	Merkle 2^{30}-tree ZK proof Bulletproofs time		R1CS constraints
		Prove	Verify	
POSEIDON-128				
BLS12-381	2:1	16.8s	1.5s	7290
	4:1	13.8s	1.65s	4500
	8:1	11s	1.4s	4050
BN254	2:1	11.2s	1.1s	7290
	4:1	9.6s	1.15s	4500
	8:1	7.4s	1s	4050
Ristretto	2:1	8.4s	0.78s	7290
	4:1	6.45s	0.72s	4500
	8:1	5.25s	0.76s	4050
SHA-256 [19]				
$GF(2^{256})$	2:1	582s	21s	762000

Table 6: PLONK performance to prove a 1-out-of-2^n-Merkle tree of arity 4.

Field	Set size	Merkle 2^n-tree ZK proof PLONK time		R1CS constraints
		Prove	Verify	
POSEIDON-128				
BLS12-381	2^{16}	3.59s	0.7ms	2400
	2^{34}	6.3s	1.55ms	5100
	2^{68}	9.9s	2.7ms	10200

over polynomial values at adjacent points.

As a result, our optimized PLONK compiler makes only $(w+11)R$ point multiplications for a single permutation call, whereas the proof consists of $(w+3)$ group elements and $2w$ field elements. This might bring a 25-40x increase in performance depending on w.

6.2.4 RedShift

RedShift [41] is a STARK-inspired proof system which works with an arbitrary set of constraints. It can be viewed as PLONK with pairing-based polynomial commitments with the trusted setup being replaced by Reed-Solomon trustless commitments. The RedShift proof is $c_\lambda \log d^2$ KB large, where d is the degree of the circuit polynomials and $c_\lambda \approx 2.5$ for a 120-bit security. Due to similarity, we can make the same optimizations as in PLONK, so that the entire Merkle tree proof requires polynomials of degree 4800 for width 5, resulting in the entire proof being around 12 KB in size. Unfortunately, no RedShift library is publicly available so far, and hence we could not measure the actual performance.

6.3 Comparison with Other Hash Algorithms

Unfortunately, no zero-knowledge system implementation contains all the primitives we want to compare with. However, for all systems we are interested it, the prover performance increases monotonically (and in practice, almost linearly) with the number of multiplications or, equivalently, with the number of R1CS constraints. We thus provide a summary of constraint counts for various hash functions in the concrete case of a Merkle tree with 2^{30} elements in Table 4. We take Blake2s and Pedersen hash estimates from [38], the SHA-256 count from Hopwood's notes[15], whereas for MiMC and *Rescue* we calculated them ourselves based on the round numbers provided in [4, 6]. The table implies that POSEIDON and *Rescue* should have the fastest provers, which is also confirmed for the STARK case [11]. However, *Rescue* has a slower performance in the non-ZK case (Table 1).

6.4 STARKs with POSEIDON$^\pi$

ZK-STARKs [9] is a proof system for the computational integrity, which is not vulnerable to quantum computers and does not use a trusted setup. STARKs operate with programs whose internal state can be represented as a set of w registers, each belonging to a binary field $\mathrm{GF}(2^n)$ or to a 2^n-subgroup $\mathbb{G}$ of a prime-order group (this is our primary case, as the scalar fields of BLS12-381 and BN254 have such a big subgroup).

The program execution is then represented as a set of T internal states. The computational integrity is defined as the set of all wT registers satisfying certain s polynomial equations (constraints) of degree d.

STARK Costs. According to [51], the number of constraints does not play a major role in the prover, verifier, or communication complexity, which are estimated as follows:

- $8w \cdot T \cdot d \cdot \log(wT)$ operations in $\mathbb{G}$ for the prover,
- a prover memory in $\Omega(w \cdot T \cdot n)$, and
- a communication (verifier time) of $n \cdot (m + \log^2(8Td))$,

where m is the maximum number of variables in a constraint polynomial.

The primitive POSEIDON$^\pi$ can be represented as such a program with few registers, a small number of steps, and low

[15] https://www.zfnd.org/zcon/0/workshop-notes/Zcon0%20Circuit%20Optimisation%20handout.pdf

degree. Following the same approach as for SNARKs in Section 6.2, we keep in registers only S-box inputs and the permutation outputs. Setting $w = t$, we get $T = R_F + \lceil R_P/t \rceil$ and $wT = tR_F + R_P$. Thus, the complexity is as follows:

- $24(tR_F + R_P) \cdot \log_2(tR_F + R_P)$ operations in $\mathbb{G}$ for the prover,
- a prover memory in $\Omega(63 \cdot (tR_F + R_P))$, and
- a communication (verifier time) of $63 \cdot (t + \log_2^2(24(tR_F + R_P)))$.

We suggest $t \in \{3,5\}$ in order to support the same Merkle tree cases as before. Thus, for our primary instance POSEIDON-128, we get an AET cost of 20540 for each permutation call for a width of 3. As we process 510 bits per call, we obtain a prover complexity of 40 operations per bit. For a width of 5 we get an AET cost of 38214, which translates to 38 operations per bit in $\mathbb{G}$.

7 Acknowledgements

This work is partially supported by the Ethereum foundation, Starkware Ltd, and IOV42 Ltd. We thank Alexander Vlasov, Lovesh Harshandani, and Carlos Perez for benchmarking POSEIDON in various environments. This work was also supported by the EUH2020 European Union's Horizon 2020 research and innovation programme (https://ec.europa.eu/programmes/horizon2020/en) under grant agreement 871473 (KRAKEN).

References

[1] C++ library for zkSNARK. https://github.com/scipr-lab/libsnark.

[2] 2019. Mike Lodder, Sovrin's principal cryptographer www.sovrin.org, private communication.

[3] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC. In *ASIACRYPT 2019*, volume 11923 of *LNCS*, pages 371–397, 2019.

[4] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 191–219, 2016.

[5] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 430–454, 2015.

[6] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. https://eprint.iacr.org/2019/426.

[7] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104. ACM, 2017.

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

[9] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO (3)*, volume 11694 of *LNCS*, pages 701–732. Springer, 2019.

[10] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796. USENIX Association, 2014.

[11] Eli Ben-Sasson, Lior Goldberg, and David Levit. Stark friendly hash – survey and recommendation. Cryptology ePrint Archive, Report 2020/948, 2020. https://eprint.iacr.org/2020/948.

[12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197, 2008.

[13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

[14] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of Oddity – New Cryptanalytic Techniques against Symmetric Primitives Optimized for Integrity Proof Systems. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *LNCS*, pages 299–328. Springer, 2020.

[15] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23, 1999.

[16] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[17] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[18] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In *FSE 2011*, volume 6733 of *LNCS*, pages 252–269, 2011.

[19] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society, 2018.

[20] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography*, volume 5443 of *LNCS*, pages 481–500. Springer, 2009.

[21] Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[22] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS*, pages 1825–1842. ACM, 2017.

[23] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768, Cham, 2020. Springer International Publishing.

[24] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

[25] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In *IMACC*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001.

[26] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[27] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.

[28] Víctor Gayoso Martínez, Luis Hernández Encinas, and Carmen Sánchez Ávila. A survey of the elliptic curve integrated encryption scheme. 2010. available at https://core.ac.uk/download/pdf/36042967.pdf.

[29] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *USENIX Security Symposium*, pages 1069–1083. USENIX Association, 2016.

[30] Lorenzo Grassi, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. *IACR Cryptol. ePrint Arch.*, 2019:458, 2019.

[31] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In *EUROCRYPT 2020*, volume 12106 of *LNCS*, pages 674–704, 2020.

[32] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. Subspace Trail Cryptanalysis and its Applications to AES. *IACR Trans. Symmetric Cryptol.*, 2016(2):192–225, 2016.

[33] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P Smart. MPC-friendly symmetric key primitives. In *CCS*, pages 430–443. ACM, 2016.

[34] Lorenzo Grassi, Christian Rechberger, and Markus Schofnegger. Weak Linear Layers in Word-Oriented Partial SPN and HADES-Like Schemes. Cryptology ePrint Archive, Report 2020/500, 2020. https://eprint.iacr.org/2020/500.

[35] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

[36] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. Practical Collision Attacks against Round-Reduced SHA-3. *Journal of Cryptology*, 33(1):228–270, 2020.

[37] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain Family of Stream Ciphers. In *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 179–190. Springer, 2008.

[38] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification: Version 2020.1.14 [overwinter+sapling+blossom+heartwood+canopy]. Technical report, Zerocoin Electric Coin Company,

2019. available at https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[39] Thomas Jakobsen and Lars R. Knudsen. The Interpolation Attack on Block Ciphers. In *FSE 1997*, volume 1267 of *LNCS*, pages 28–40, 1997.

[40] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010.

[41] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. Redshift: Transparent snarks from list polynomial commitment iops. Cryptology ePrint Archive, Report 2019/1400, 2019. https://eprint.iacr.org/2019/1400.

[42] Nathan Keller and Asaf Rosemarin. Mind the Middle Layer: The HADES Design Strategy Revisited. Cryptology ePrint Archive, Report 2020/179, 2020. https://eprint.iacr.org/2020/179.

[43] Lars R. Knudsen. Truncated and Higher Order Differentials. In *FSE 1994*, volume 1008 of *LNCS*, pages 196–211, 1994.

[44] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 126–143, 2009.

[45] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 1978.

[46] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2111–2128. ACM, 2019. URL: https://doi.org/10.1145/3319535.3339817, doi:10.1145/3319535.3339817.

[47] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In *EUROCRYPT 1993*, volume 765 of *LNCS*, pages 386–397, 1993.

[48] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In *FSE 2009*, volume 5665 of *LNCS*, pages 260–276, 2009.

[49] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[50] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In *Fast Software Encryption – FSE 1996*, volume 1039 of *LNCS*, pages 99–111. Springer, 1996.

[51] StarkWare Industries Ltd. The complexity of STARK-friendly cryptographic primitives. Private communication, 2018.

[52] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper.(2014), 2014.

[53] A. M. Youssef, S. Mister, and S. E. Tavares. On the Design of Linear Transformations for Substitution Permutation Encryption Networks. In *School of Computer Science, Carleton University*, pages 40–48, 1997.