On Efficient and Secure Compression Functions for Arithmetization-Oriented Hashing

Andreeva, Elena; Bhattacharyya, Rishiraj; Roy, Arnab; Trevisani, Stefano

DOI:
10.1109/CSF61375.2024.00032

On Efficient and Secure Compression Functions for Arithmetization-Oriented Hashing

Elena Andreeva
Research Unit of Security and Privacy
TU Wien
Favoritenstraße 9–11, 1040 Vienna, Austria
elena.andreeva@tuwien.ac.at

Rishiraj Bhattacharyya
School of Computer Science
University of Birmingham
Edgbaston, B15 2TT Birmingham, United Kingdom
r.bhattacharyya@bham.ac.uk

Arnab Roy
Security and Privacy Lab
University of Innsbruck
Technikerstraße 21A, 6020 Innsbruck, Austria
arnab.roy@uibk.ac.at

Stefano Trevisani
Research Unit of Security and Privacy
TU Wien
Favoritenstraße 9–11, 1040 Vienna, Austria
stefano.trevisani@tuwien.ac.at

***Abstract*—ZK-SNARKs, a fundamental component of privacy-oriented payment systems, identity protocols, or anonymous voting systems, are advanced cryptographic protocols for verifiable computation: modern SNARKs allow to encode the invariants of a program, expressed as an arithmetic circuit, in an appropriate constraint language from which short, zero-knowledge proofs for correct computations can be constructed.**

One of the most important computations that is run through SNARK systems is the verification of Merkle tree (MT) opening proofs, which relies on the evaluation of a fixed-input-length (FIL) cryptographic compression function over binary MTs. As classical, bit-oriented hash functions like SHA-2 are not compactly representable in SNARK frameworks, Arithmetization-Oriented (AO) cryptographic designs have emerged as an alternative, efficient solution.

Today, the majority of AO compression functions are built from permutation-based hashing modes, such as Sponge. While this approach allows cost savings, compared to blockcipher-based modes, as it does not require key-scheduling, AO blockcipher schedulers are often cheap to compute. Furthermore, classical bit-oriented cryptography has long studied how to construct provably secure compression functions from blockciphers, following the Preneel-Govaerts-Vandewalle (PGV) framework. The potential efficiency gains together with the strong provable security foundations in the classic setting, motivate the study of AO blockcipher-based compression functions.

In this work, we propose AO PGV-LC **and** PGV-ELC**, two AO blockcipher-based FIL compression modes inspired by and extending the classical PGV approach, offering flexible input and output sizes and coming with provable security guarantees in the AO setting. We prove the collision and preimage resistance in the ideal cipher model, and give bounds for collision and opening resistance over MTs of arbitrary arity.**

We compare experimentally the AO PGV-ELC **mode over the** HADES **blockcipher with its popular and widely adopted Sponge instantiation,** POSEIDON**, and its improved variant** POSEIDON2**. Our resulting constructions are up to** $3\times$ **faster than** POSEIDON **and** $2\times$ **faster than** POSEIDON2 **in native x86 execution, and up to** 50% **faster in the Groth16 SNARK framework.**

Finally, we study the benefits of using MTs of arity wider than two, proposing a new strategy to obtain a compact R1CS constraint system in such case. In fact, by combining an efficient parametrization of the HADES **blockcipher over the** PGV-ELC **mode, together with an optimal choice of the MT arity, we measured an improvement of up to** $9\times$ **in native MT construction time, and up to** $2.5\times$ **in proof generation time, compared to** POSEIDON **over binary MTs.**

***Index Terms*—Hash function, Block cipher, Arithmetization-Oriented, Merkle tree, Zero-Knowledge, SNARK**

I. INTRODUCTION

Zero-Knowledge Frameworks and Hash Functions.: Zero-Knowledge Proof (ZKP) systems [1], [2] are advanced cryptographic protocols which allow a *prover* to convince a *verifier* about the solvability of some problem instance, yet without having to reveal the solution. Nowadays, general-purpose, zero-knowledge *Succinct Non-interactive ARgument of Knowledge* (ZK-SNARK) systems [3]–[8], allow the prover to build short proofs for the *computational integrity* of some bounded algorithm, which can be efficiently checked by any number of verifiers in a non-interactive way. This process involves an *arithmetization* step, where the invariants of the algorithm are encoded in a set of *algebraic constraints* [7], [9], [10]. ZK-SNARKs are extensively used for privacy-preserving payment systems [11]–[13], anonymous distributed storage systems [14], authenticated machine learning and image processing [15], [16], anonymous voting [17], to name a few [18].

Cryptographic hash functions are fundamental building blocks for SNARKs and their *security* and *efficiency*. Hash functions compress either fixed-input-lengths (FIL, a.k.a. compression functions) or variable-input-lengths (VIL) and where necessary we make this distinction. In SNARK systems hash functions are used for a variety of tasks ranging from data integrity checks, commitment schemes to setting up non-interactive proofs with the Fiat-Shamir transform [19]. More specifically, two of the most prominent use cases of hash functions in ZK-SNARKs are:

- Efficient and secure zero-knowledge set membership proofs based on the Merkle Tree (MT) hash accumulator, a fundamental part of privacy-preserving blockchains like Zcash and Filecoin [11], [14]. In this setting, users $\mathcal{U}_1, \ldots, \mathcal{U}_m$ insert their values $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ (e.g. unspent transaction in Zcash) in the MT accumulator $\mathcal{T}$. The MT root value is published. To claim ownership of $\boldsymbol{v}_j$, the user must prove knowledge of the value. The ZK proof is a proof of computation for the *opening proof* (Def.10) for the MT. The latter is a sequence of preimages of FIL compression functions for values along the MT path that are combined to obtain the MT root. In these proof systems, the size of the *hash function arithmetic circuits*, i.e. its *multiplicative complexity* over a prime field $\mathbb{F}_p$, is the dominant cost metric, rather than its plain or "native" performance.

- In recursive SNARK frameworks, such as FRACTAL [20] or Halo Infinite [21], the entire computation can be viewed as consecutive applications of a chain of functions $f_1, \ldots, f_m$ on some state $\boldsymbol{x}_0$, where each f_i leads to an intermediate result $\boldsymbol{x}_i$, and a proof of correctness must be handed to the next prover (who computes f_{i+1}). Hence, the prover $\mathcal{P}_i$ observes of its own computation $f_i(\boldsymbol{x}_{i-1})$ and a proof of correctness from the previous prover. The verifier $\mathcal{V}_i$ checks the correctness of f_{i-1} from the corresponding proof π_{i-1}. In proof systems where the (inner) commitment for π_{i-1} consists in an MT opening, the verifier $\mathcal{V}_i$ executes again the FIL compression function underlying the MT. The efficiency of such applications critically depends on the minimization of both the *hash function circuits size* and *native running time*.

In both settings, standard or classical choices of hash functions (e.g. SHA-2 or SHA-3 [22]) are the bottleneck of the computation since their arithmetization requires a huge number of algebraic constraints, slowing down significantly the proof generation process. The efficiency of SNARK systems depend on the size, and particularly on the *multiplicative complexity*, of the *arithmetic circuit* of the hash function and that over a large prime field $\mathbb{F}_p$. Hence, to optimize performance in the SNARK setting, a hash function design should optimize both its *arithmetic circuit size* and its *multiplicative complexity*. Additionally, when the hash function is used in the Merkle tree, it accepts *FIL inputs*, meaning that taking into account excessive design requirements (e.g. padding or iterating for VIL hash functions) incurs further complexity.

Arithmetization-Oriented Hash Functions.: Driven by the efficiency requirements of modern ZK proof systems, a number of *so-called Arithmetization-Oriented* (AO) hash and compression functions were proposed in the literature. The AO hash function main features can be summarized as follows:

- They are defined over *large prime fields* $\mathbb{F}_p$, where usually $p \approx 2^{256}$, and $\mathbb{F}_p$ is the scalar fields of some elliptic curve (Table VII).
- They are constructed in a *modular* way, by applying a *permutation-based mode of operation* to either a keyless permutation [23]–[25] or a fixed-key blockcipher [26]–[31] (realizing also a permutation). The underlying permutation is parameterized by the size of the input ($\in \mathbb{F}_p^n$), while the hash function is parameterized by the size of the output ($\in \mathbb{F}_p^l$, with $l < n$).
- They aim at reducing the arithmetization cost over ZK-SNARK frameworks. Hash functions that are devised for arbitrary choices of field and proof system include MiMChash [26], GMiMCHash [32], *Rescue Prime* [27], POSEIDON [28], POSEIDON2 [30], Anemoi [24], GRIFFIN [25], and Arion [31], [33]. Hash functions that are devised for only specific proof systems and choices of p include Reinforced Concrete [23], Tip5 [34], and Monolith [35]: these designs aim at improving the native performance by using lookup tables, but require a SNARK system which supports lookup constraints [20], [36].

Implementation Considerations.: In the landscape of ZK-SNARK systems, different *constraint languages* are used in order to represent the invariants of arithmetic circuits. One of the most popular arithmetization techniques are R1CS constraint systems, where the computation is constrained via a system of *bilinear equations* of the kind $\boldsymbol{ax} \times \boldsymbol{by} = \boldsymbol{cz}$. The size of the constraint system, and hence the cost of arithmetization, mainly depends on the *multiplicative complexity*, i.e. the number of multiplication gates, of the arithmetic circuit [5]. R1CS arithmetization is adopted by all implementations of the famous Groth16 [6] SNARK framework, such as libsnark [37], bellman [38], arkworks [39], and Circom [40], [41], among others. Given that one of the most important applications of AO hash functions are the computation of Merkle tree commitments and openings, the focus should not only be on the cost of the underlying FIL compression function. Though the *binary* Merkle tree is undoubtedly a very widely adopted structure, it might not necessarily be used in an optimal way in this setting. To the best of our knowledge, the direction that has been largely unexplored is establishing the trade-offs and potential improvements in both native and SNARK performance by combining a larger and possibly more expensive FIL hash function with a MT tree of *wider arity* and hence lower height (for the same number of leaves).

Modular hash function design.: The modular approach is paramount in cryptography. It allows for: 1. Provable security arguments up to the soundness of the building blocks; 2. Reduces complexity by shifting the focus of the analysis to the usually simpler and smaller scale building blocks; 3. Allows for off-the-shelf replacements of the building blocks. We outline the most common modular design strategy used thus far for FIL compression functions for ZK proof systems. To compress inputs from $\mathbb{F}_p^r$ to $\mathbb{F}_p^l$, for some $l < r$, a secure permutation $\pi \colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ in a Sponge mode [42] is called in the so-called *absorb* step, where the permutation size is $n > r$. Then, vectors of field elements are compressed via:

$$\mathrm{Sponge}(\boldsymbol{x}) = \mathrm{Tr}_l\big(\pi\big(\boldsymbol{x} \parallel \mathbf{0}^{n-r}\big)\big)$$

Here $\parallel$ denotes vector concatenation, Tr_l the vector truncation up to the lth component, and $\mathbf{0}^{n-r}$ the zero vector over $\mathbb{F}_p^{n-r}$.

A number of AO hash functions [26]–[28], [31], [32] rely on the Sponge mode of hashing to achieve fixed-input-length (FIL) compression. Although the Sponge mode [42] is well-defined for inputs over any arbitrary group, its formal random-oracle indifferentiability [43] proof was *only* carried out for the special case of $\mathbb{F}_2$ [44]. This fact leaves a gap in the formal security argument of AO hash functions relying on the Sponge mode for inputs over $\mathbb{F}_p$.

Two alternative FIL compression modes, Trunc and `Jive`, were proposed respectively in [25] and [24]: in both cases, the permutation size is $n = r$, and vectors are compressed through:

$$\text{Trunc}(\boldsymbol{x}) = \text{Tr}_l(\boldsymbol{y}) \qquad \texttt{Jive}(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} y_i$$

where $\boldsymbol{y} = \pi(\boldsymbol{x}) + \boldsymbol{x}$. Compared to the Sponge mode, both Trunc and `Jive` require a smaller permutation, improving efficiency. Although the underlying instantiations were proposed as a keyless permutation for both Trunc and `Jive` modes, in [25], the security of both modes was linked to the Davies-Meyer construction [45], by viewing the (keyless) permutation $\pi(\boldsymbol{x})$ as a fixed-key blockcipher $E_{\boldsymbol{v}}(\boldsymbol{x})$. And no formal security proof was provided. However, the Davies-Meyer proof argument [46], [47] does not apply to permutations and applies *only* for the boolean field $\mathbb{F}_2$.

Although all major AO hash functions employ internally a permutation, these permutations happen to be most commonly obtained from a *readily designed AO blockcipher* $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{l}$. The key of the blockcipher is set to a fixed constant to get a permutation. This is for example the case in one of the most popular AO hash functions, POSEIDON, and its newer version POSEIDON2: the former has been deployed in many real-world systems, including Filecoin [14], Loopring [48], Zcash [49], and it has been proposed to be added in the Ethereum VM [50]. Both constructions are based on the HADES cipher [51] by fixing the key in the cipher and applying the Sponge mode. While in classical bit-oriented cryptography key-scheduling for the blockcipher is often an expensive operation (e.g. in AES [52], [53]), for AO blockciphers like MiMC, GMiMC and HADES, the key scheduling algorithm is an affine function whose computational cost is *cheap* both in native execution and in the SNARK circuit.

AO cryptography and design goals.: The field of AO-based cryptography is less than a decade old and still in its naissance. At present, novel designs approaches for AO hashing are emerging, typically coming with contemporary evidence of lack of cryptanalytic flaws, and efficiency benchmarks. Cryptanalysis techniques for such constructions are constantly evolving [54]–[56], often exposing important weaknesses in the design approaches [57]–[60].

A cryptographic design however stands the test of time upon evidence of sound, simple and verifiable security arguments (cryptanalysis and proofs of security), significant efficiency advantages and flexible design features. A flexibility design feature of AO hashing is for example the ability to handle different hash function input and output sizes, or internal building block sizes, a feature currently not present in existing permutation-based designs.

The most popular classical approach to designing a provably secure FIL compression function over binary inputs is the Preneel-Govaerts-Vandewalle (PGV) framework [61]. Its compositional simplicity has established it a foundational design approach for a number of classical hash functions, including RIPEMD-160 and SHA-2 [22], [62]. PGV designs have been provably investigated by Black et al. [47] in the *ideal cipher model*. In PGV, the inputs are compressed via a secure blockcipher $E\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ as follows:

$$\text{PGV}(\boldsymbol{x} \parallel \boldsymbol{y}) = E_{\boldsymbol{a}}(\boldsymbol{b}) + \boldsymbol{c}$$

where $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}+\boldsymbol{y}, \boldsymbol{v}\}$, for some constant value $\boldsymbol{v}$ (for example, if $\boldsymbol{a} = \boldsymbol{y}$, and $\boldsymbol{b} = \boldsymbol{c} = \boldsymbol{x}$, one obtains the previously mentioned Davies-Meyer mode). While the PGV framework has been integral to a number of classical hash function design, its security does not apply in the AO setting over large *prime fields* $\mathbb{F}_p$.

Taking in consideration the largely unexplored area of AO blockcipher modular compositionality towards FIL compression function designs and their performance, in this work we aim to answer the following main questions:

1) Can we design provably secure and simple AO and FIL compression functions for ZK proof systems over arbitrary prime fields $\mathbb{F}_p$ based on AO blockciphers?
2) Can such FIL compression functions have *flexibility* with respect to input, output and internal primitive sizes?
3) Do these blockcipher-based functions, both theoretically and experimentally, offer significant improvements with respect to comparable existing (for the same AO blockcipher with a fixed key) permutation-based designs?
4) Is it possible to exploit the relation between the cost of the FIL compression function over its state size and of the Merkle tree opening proof over its arity to further optimize the performance?

Our Contributions

In this work we make the following contributions:

1) AO SYNTAX AND SECURITY DEFINITIONS. Towards sound security analysis, in Section II, we *adapt the classical syntax* for blockcipher and permutation-based VIL/FIL hash functions to the AO setting over arbitrary prime fields $\mathbb{F}_p$. Additionally, we *tailor the formal security definitions* of *collision and preimage resistance* for VIL/FIL hash functions to the AO context, and similarly capture the notion of *opening resistance* notion over Merkle trees of arbitrary arity over $\mathbb{F}_p$.
2) NEW AO FIL COMPRESSION FUNCTIONS STRATEGIES. In Section III, we propose the AO blockcipher-based AO FIL compression function strategy PGV-LC. PGV-LC is flexible as it is defined to work on inputs of dimension $\kappa' = \kappa$ and $n' = n$, a blockcipher $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$, and an $n \times l$ post-processing matrix $\boldsymbol{R}$, where l is some

desired (flexible) output size, the compression is carried out as (also see Figure 1):

$$\text{PGV-LC}(\boldsymbol{x} \parallel \boldsymbol{y}) = \boldsymbol{R} \cdot (E_{\boldsymbol{x}}(\boldsymbol{y}) + \boldsymbol{y})$$

The PGV-LC mode generalizes to the AO setting the compression functions in the PGV framework, and also encompasses the Trunc and `Jive` modes, if interpreted as in [25], by giving a fully linear-algebraic characterization. From PGV-LC, we derive the next FIL compression function instantiation strategy PGV-ELC. PGV-ELC offers further flexibility by allowing the input dimensions to be smaller than the blockcipher dimensions by applying additional pre-processing matrices $\boldsymbol{P}$, $\boldsymbol{K}$, and $\boldsymbol{F}$ (see Figure 2).

3) SECURITY PROOFS FOR PGV-LC AND PGV-ELC. In Section IV, we prove the security of PGV-LC and PGV-ELC in the *ideal cipher model*. We show that there are natural classes of the linear transformations involved in the compression for which both modes are collision resistant up to $(q^2+q)/(p^l-q)$ queries to the underlying blockcipher, and preimage resistant up to $q/(p^l-q)$ queries.
For the target ZK proof systems applications, we come with a proof that reduces the collision and opening proof forgery resistance of t-ary AO Merkle trees to the collision and preimage resistance of the underlying compression function. Our proofs are generic in nature and enable secure instantiations with sound AO blockciphers, namely, they allow the bulk of cryptanalysis to be shifted to the underlying blockcipher.

4) OPTIMIZATIONS AND EXPERIMENTS. In Section V, we consider the widely adopted hash function POSEIDON [14], [28], [48], [50], and its newer variant, POSEIDON2 [30], which are both based on a fixed-key instantiation of the HADES [51] blockcipher. We compare (Sponge) POSEIDON and POSEIDON2-Trunc with similar PGV-ELC instantiations of HADES, which we call POSEIDON-DM and POSEIDON2-DM, respectively.
First, we present an optimized way to synthesize R1CS constraint systems for arbitrary arity MT opening proofs, which, when applied to POSEIDON-DM, already offers a 5–15% improvement in the number of constraints compared to alternative techniques (Table II).
In R1CS-based ZK-SNARK systems, POSEIDON-DM and POSEIDON2-DM require up to 50% less R1CS constraints than POSEIDON and POSEIDON2 (Table III); additionally, by exploiting our optimized Merkle tree constraint system, optimal arity choices for POSEIDON and POSEIDON2-DM are ≈ 2.5× faster than POSEIDON and POSEIDON2 over binary trees in the Groth16 [6] framework (Table IV and fig. 3).
For native computations, POSEIDON-DM is up to 3× faster than POSEIDON, while POSEIDON2-DM is up to 2× faster than POSEIDON2. For the optimal arity choice, when constructing Merkle trees, POSEIDON2-DM is 2.5× faster than POSEIDON2 over binary trees, and almost 9× faster than POSEIDON over binary trees, which is the most popular instance used in real-world applications (Table V and fig. 4). When parallelizing the MT construction, we noticed that all compression functions scale similarly, with the size of the underlying prime field being the most relevant bound (Figure 6).

II. PRELIMINARIES

A. Notations and Definitions

Arithmetization-Oriented cryptography is concerned with the design of cryptographic algorithms that manipulate elements of finite algebraic structures (e.g. fields and vector spaces), rather than strings of bits.

Given a set S of cardinality $|S|$, let $S^* = \bigcup_{i\in\mathbb{N}} S^i$ denote the *Kleene's closure* of S, and let S^ω denote the set of infinite-length tuples made from elements of S. Given a prime number p, let $\mathbb{F}_p$ the *finite prime field* of *order* $|\mathbb{F}_p| = p$ and *characteristic* $\text{char}(\mathbb{F}_p) = p$, with canonical addition and multiplication modulo p. We will consider p to be odd, and typically 'large' (say, $p > 2^{64}$). We denote with $\mathbb{F}_p^n$ the standard n-dimensional *vector space* of *column vectors* over $\mathbb{F}_p$, with standard addition and scalar product. Similarly, $\mathbb{F}_p^{n\times m}$ is the standard $(n \times m)$-dimensional matrix space over $\mathbb{F}_p$. Scalars are denoted with lowercase letters $a, b, c, \ldots$, vectors with bold lowercase letters $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$, and matrices with bold uppercase letters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$. We denote with $\boldsymbol{I}^{n\times m}$ the pseudo-identity matrix whose entries in the main diagonal have value 1 while all other entries have value 0. The transpose of a vector $\boldsymbol{a}$ (respectively a matrix $\boldsymbol{A}$) is denoted with $\boldsymbol{a}^\intercal$ (respectively $\boldsymbol{A}^\intercal$).

Most of the following definitions are well-known in classical symmetric cryptography over $\mathbb{F}_{2^n}$. We lift them over $\mathbb{F}_p$ to facilitate the discussion on AO modes.

Definition 1 (AO blockcipher)**.** Given some $\kappa, n \in \mathbb{N}$, and a prime field $\mathbb{F}_p$, a κ-n-elements AO blockcipher over $\mathbb{F}_p$ is a function:

$$E(\boldsymbol{k}, \boldsymbol{x}) \colon \mathbb{F}_p^\kappa \times \mathbb{F}_p^n \to \mathbb{F}_p^n$$

which is a permutation on $\boldsymbol{x}$ for every possible choice of $\boldsymbol{k}$. An AO blockcipher family $\{E_{\boldsymbol{k}}\}$ is the collection of all permutations $E_{\boldsymbol{k}}(\boldsymbol{x})$ obtained by partial application of $\boldsymbol{k}$, and $\{E_{\boldsymbol{k}}^{-1}\}$ is the collection of all their inverses.

When κ is left unspecified, we implicitly assume $\kappa = n$. Following a standard abuse of notation, we will often write E to mean $\{E_{\boldsymbol{k}}\}$ and E^{-1} to mean $\{E_{\boldsymbol{k}}^{-1}\}$.

Definition 2 (AO compression function)**.** Given some $m, n \in \mathbb{N}$, with $m > n$, and a prime field $\mathbb{F}_p$, an m-n-elements AO compression function over $\mathbb{F}_p$ is any function with signature:

$$C(\boldsymbol{x}) \colon \mathbb{F}_p^m \to \mathbb{F}_p^n$$

For ease of discussion, we may describe an ml-n-elements compression function in terms of multiple arguments $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{F}_p^l$ rather than in terms of one single argument

$\boldsymbol{x} \in \mathbb{F}_p^{ml}$, and we equivalently write $m{:}n$ to denote m-n-elements compression. Compression functions are also known as Fixed-Input-Length (FIL) hash functions.

Definition 3 (AO hash function)**.** Given some $n \in \mathbb{N}$, and a prime field $\mathbb{F}_p$, an n-elements AO hash function over $\mathbb{F}_p$ is any function with signature:

$$H(M)\colon (\mathbb{F}_p)^* \to \mathbb{F}_p^n$$

Variable-Input-Length (VIL) n-elements hash functions are generally built on top of some m-n FIL compression function together with an l-elements padding function of the kind:

$$\mathrm{Pad}(M)\colon (\mathbb{F}_p)^* \to \left(\mathbb{F}_p^l\right)^*$$

where l is an appropriate multiple of m which depends on the structure of the hash function itself. It is extremely important to use well-behaved padding functions, even more so in AO cryptography where there does not exist a bijective mapping between elements of $\mathbb{F}_p$ and bit-strings of a certain length (except when $p = 2$); however, as this work is mostly concerned with FIL compression, we assume that such a padding function is available.

B. AO Modes of Operation

Directly devising secure cryptographic algorithms is not an easy task; the standard approach is to directly design relatively simple primitives, such as (unkeyed) permutations or blockciphers, and then compose them in a black-box manner through a *mode of operation* to obtain more advanced functionalities.

A famous family of modes to build secure compression and hash functions from blockciphers are the PGV modes [61]. The PGV modes are tightly related to the Merkle-Damgård (MD) mode of hashing [63], in that they generalize well-known modes like Davies-Meyer [45], Matyas-Meyer-Oseas [64], or Miyaguchi-Preneel [65], [66], and are hence defined with respect to MD inputs: a message block, a chaining variable and an initialization value (IV). While classical PGV modes are defined over bit-strings, it is easy to adapt their definition to the AO context: we will refer to these modes explicitly as AO PGV-MD modes.

Definition 4 (AO PGV-MD modes)**.** Given an n-elements blockcipher E over some prime field $\mathbb{F}_p$, an initialization value $\boldsymbol{v} \in \mathbb{F}_p^n$, a chaining value $\boldsymbol{h}_{i-1}$ such that $\boldsymbol{h}_0 = \boldsymbol{v}$, the AO PGV-MD modes of E are all the compression functions of the kind:

$$\boldsymbol{h}_i = C_{E,\boldsymbol{v}}(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i) = E_{\boldsymbol{a}}(\boldsymbol{b}) + \boldsymbol{c}$$

where $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \{\boldsymbol{x}_i, \boldsymbol{h}_{i-1}, \boldsymbol{v}, \boldsymbol{x}_i + \boldsymbol{h}_{i-1}\}$.

A more recent approach to build secure FIL/VIL hash functions is the Sponge mode [42]. Rather than using a blockcipher as the underlying primitive, the Sponge mode operates over an unkeyed permutation.

Definition 5 (AO Sponge mode)**.** Given an n-elements permutation π over some prime field $\mathbb{F}_p$, a rate $r < n$, and a padding function $\mathrm{Pad}\colon (\mathbb{F}_p)^* \to \left(\mathbb{F}_p^r\right)^*$, let the Sponge iteration function with rate r of π be:

$$s_i(M)\colon \left(\mathbb{F}_p^r\right)^* \to \mathbb{F}_p^n = \begin{cases} \mathbf{0} & i = 0 \\ \pi(s_{i-1}(M) + \boldsymbol{m}_i) & 1 \le i \le |M| \\ \pi(s_{i-1}(M)) & i > |M| \end{cases}$$

where the vectors $\boldsymbol{m}_i \in \mathbb{F}_p^r$ are implicitly naturally embedded in $\mathbb{F}_p^n$. Then, the Sponge mode of π with rate r is the function:

$$\tilde{S}_\pi(M)\colon \left(\mathbb{F}_p^r\right)^* \to \left(\mathbb{F}_p^r\right)^\omega = s_{|M|}(M) \parallel s_{|M|+1}(M) \parallel \ldots$$

and the Sponge mode of π with rate r and padding function Pad is the function:

$$S_{\mathrm{Pad},\pi}(M)\colon (\mathbb{F}_p)^* \to \left(\mathbb{F}_p^r\right)^\omega = \tilde{S}_\pi(\mathrm{Pad}(M))$$

The quantity $c = n - r$ is called the *capacity* of the Sponge. A Sponge construction is an *extendable output function* (XOF) [67]: we can truncate its output to obtain a hash function, and fix the input length to obtain a compression function.

An alternative to sequential modes like MD and Sponge is Merkle tree (MT) hashing [68], [69], a way of compressing message blocks in a parallel fashion. Differently from both Sponge and MD, the MT hashing uses a FIL compression function as the underlying primitive.

Definition 6 (AO Merkle tree mode)**.** Given some $l, t \in \mathbb{N}$, a tl-l-elements compression function C over some prime field $\mathbb{F}_p$, and a message $M \in \left(\mathbb{F}_p^l\right)^*$ such that $\exists h \in \mathbb{N}\colon |M| = t^h$, let the Merkle tree over C and M be the t-ary tree $\mathcal{T}_{C,M}$ of height h, containing $n = |\mathcal{T}_{C,M}| = \frac{t^{h+1}-1}{t-1}$ nodes $\boldsymbol{\nu}_0, \ldots, \boldsymbol{\nu}_{n-1} \in \mathbb{F}_p^l$ ordered in a top-down left-to-right manner, and rooted in $\boldsymbol{\nu}_0$, such that $\forall i < n$:

$$\boldsymbol{\nu}_i = \begin{cases} C(\boldsymbol{\nu}_{ti+1}, \ldots, \boldsymbol{\nu}_{ti+t}) & 0 \le i < n - t^h \\ \boldsymbol{m}_{i+1-(n-t^h)} & n - t^h \le i < n \end{cases}$$

Given a function $\mathrm{Pad}\colon (\mathbb{F}_p)^* \to \left(\mathbb{F}_p^l\right)^*$ such that $|\mathrm{Pad}(M)| = t^h$ for some $h \ge 1$, the Merkle tree mode of C with padding function Pad is the hash function:

$$H_{C,\mathrm{Pad}}(M)\colon (\mathbb{F}_p)^* \to \mathbb{F}_p^l = \boldsymbol{\nu}_0$$

Merkle trees are widely used in many applications, such as version control systems [70], P2P networks [71], [72], and database systems [73], [74]. In particular, they play a crucial role in blockchains [75], [76] to create proofs of membership.

The most common approach to instantiate the Merkle tree's underlying compression function is with a Sponge-mode permutation. Recently, new FIL permutation-based compression modes have been proposed to replace the Sponge in this scenario [24], [25], [30]. In particular, we are interested in the Trunc mode used by POSEIDON2.

Definition 7 (Trunc mode)**.** Given an n-elements permutation π over some prime field $\mathbb{F}_p$, and some $l < n$, the Trunc_l mode of π is the compression function:

$$C(\boldsymbol{x})\colon \mathbb{F}_p^n \to \mathbb{F}_p^l = \boldsymbol{I}^{l \times n} \cdot \pi(\boldsymbol{x})$$

Algorithm 1 The q-queries ideal blockcipher oracle: for every choice of $\boldsymbol{k} \in \mathbb{F}_p^\kappa$, $E_{\boldsymbol{k}}$ is a random permutation over $\mathbb{F}_p^n$ with inverse $E_{\boldsymbol{k}}^{-1}$. After being queried q times, the oracle 'shuts-down'.

```
function $\mathcal{E}_{E,q}(\boldsymbol{k}, \boldsymbol{m}, b)$
    static $i \leftarrow 0$
    if $i \geq q$ then return $\perp$
    $i \leftarrow i + 1$
    if $b = 0$ then return $E_{\boldsymbol{k}}(\boldsymbol{m})$
    return $E_{\boldsymbol{k}}^{-1}(\boldsymbol{m})$
```

C. Security Notions

In order to study the cryptographic constructions of interest, we must first formalize the relevant security notions that we target. We denote with $x \xleftarrow{\$} S$ the experiment of sampling x independently and uniformly at random from some finite set S; additionally, we let $\mathsf{Block}(p, \kappa, n)$ be the set of all κ-n-elements blockciphers over $\mathbb{F}_p$.

Remark 1. Our results will be given for the *ideal AO blockcipher model*, where we assume that the blockcipher used by blockcipher-based modes is instantiated by $E \xleftarrow{\$} \mathsf{Block}(p, \kappa, n)$. The *adversary* is an information theoretical (computationally unbounded) randomized algorithm $\mathcal{A}$ with query access to the oracle $\mathcal{E}_{E,q}$, denoted $\mathcal{A}^{\mathcal{E}_{E,q}}$, which answers to at most q queries to before 'shutting down'. A description of the oracle's behaviour is given in Algorithm 1. When E and q are clear from the context, we may omit them from the subscript.

Definition 8 (COMP-COL advantage)**.** Given an m-n-elements blockcipher-based compression function C_E over some prime field $\mathbb{F}_p$, the collision advantage of an adversary $\mathcal{A}$ with q queries against C_E, denoted $\mathbf{Adv}_{C_E}^{\text{COMP-COL}}(\mathcal{A}, q)$, is equal to:

$$\Pr\left[(\boldsymbol{y}, \boldsymbol{y}') \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_{E,q}}() \colon \boldsymbol{y} \neq \boldsymbol{y}' \wedge C_E(\boldsymbol{y}) = C_E(\boldsymbol{y}')\right]$$

Definition 9 (COMP-PRE advantage)**.** Given an m-n-elements blockcipher-based compression function C_E over some prime field $\mathbb{F}_p$, the preimage advantage of an adversary $\mathcal{A}$ with q queries against C_E, denoted $\mathbf{Adv}_{C_E}^{\text{COMP-PRE}}(\mathcal{A}, q)$, is equal to:

$$\Pr\left[\boldsymbol{y} \xleftarrow{\$} \mathbb{F}_p^n, \boldsymbol{x} \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_{E,q}}(\boldsymbol{y}) \colon C_E(\boldsymbol{x}) = \boldsymbol{y}\right]$$

Similar collision and preimage advantage functions $\mathbf{Adv}^{\text{HASH-COL}}$ and $\mathbf{Adv}^{\text{HASH-PRE}}$ can be defined for hash functions. A more comprehensive treatment of advantage functions and the security properties of hash functions can be found in [77].

Now suppose that we are given a hash function H together with some digest $\boldsymbol{h} = H(M)$, for some unknown message M, and we wish to check whether a given message $M' = M$. We can do so by comparing $H(M')$ with $\boldsymbol{h}$: if the range of H is large enough, and H is both collision and preimage resistant, the check should succeed for some message $M' \neq M$ only with negligible probability, even if a potential forger has knowledge of both H and M. More generally, we can have so-called *opening proof systems*, where one party, called the *proof generator* $\mathcal{G}$, is given a message M together with an index i, and has to synthesize what essentially is a proof of membership π for $\boldsymbol{m}_i$. Then, a second party, the *proof verifier* $\mathcal{V}$, given only π and the hash of the original message, has to establish whether $\boldsymbol{m}_i$ did actually belong to M. More formally:

Definition 10 (Opening proof system)**.** Given an n-element hash function H over some prime field $\mathbb{F}_p$, an opening proof system over H is a pair of algorithms $(\mathcal{G}, \mathcal{V})_H$ such that, for any message $M \in (\mathbb{F}_p)^*$, it holds that:

$$\forall i \leq |M| \colon \mathcal{V}(\mathcal{G}(M, i), H(M)) = \top$$

In order to guarantee statistical soundness of an opening proof system, it must be hard for an attacker to forge an invalid proof, i.e. a proof of membership for some message block $\tilde{\boldsymbol{m}} \notin M$ that can fool the verifier:

Definition 11 (OPENING advantage)**.** Given an opening proof system $(\mathcal{G}, \mathcal{V})$ over some n-element blockcipher-based AO hash function H_E with underlying field $\mathbb{F}_p$, and given $M \xleftarrow{\$} (\mathbb{F}_p)^*$, the opening proof advantage of an adversary $\mathcal{A}$ with q queries against $(\mathcal{G}, \mathcal{V})$, denoted $\mathbf{Adv}_{(\mathcal{G},\mathcal{V})}^{\text{OPENING}}(\mathcal{A}, q)$, is equal to:

$$\Pr\left[\tilde{\pi} \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_{E,q}}(M) \colon \forall i \colon \tilde{\pi} \neq \mathcal{G}(M, i) \wedge \mathcal{V}(\tilde{\pi}, H_E(M)) = \top\right]$$

Given some advantage function $\mathbf{Adv}(\mathcal{A}, q)$, we let $\mathbf{Adv}(q)$ be the maximum advantage achievable by any adversary $\mathcal{A}$: $\mathbf{Adv}(q) = \max_{\mathcal{A}}\{\mathbf{Adv}(\mathcal{A}, q)\}$.

III. Two new modes of compression

Using the PGV modes design as a starting point, we extract the underlying FIL compression function, detaching it from the MD paradigm. In order to have more flexibility on the output size, we introduce an additional *linear combination* at the end of the construction, obtaining the AO PGV-LC compression mode:

Definition 12 (AO PGV-LC mode)**.** Given a κ-n-elements blockcipher E over some prime field $\mathbb{F}_p$, an output size $l \leq n$, and a right invertible reduction matrix $\boldsymbol{R} \in \mathbb{F}_p^{l \times n}$, the AO PGV-LC mode of E is the compression function:

$$C_{E,\boldsymbol{R}}(\boldsymbol{x}, \boldsymbol{y}) \colon \mathbb{F}_p^\kappa \times \mathbb{F}_p^n \to \mathbb{F}_p^l = \boldsymbol{R}(E_{\boldsymbol{x}}(\boldsymbol{y}) + \boldsymbol{y})$$

The right-invertibility property of the matrix $\boldsymbol{R}$, as we will see in Section IV, is required in order to have a secure compression. Note that when $l = n$ and $\boldsymbol{R} = \boldsymbol{I}^{n \times n}$, then our construction collapses precisely in the compression mode underlying the Davies-Meyer and the Matyas-Meyer-Oseas iterated compression functions. A visual representation of the new mode is given in Figure 1.

Based on the proposed mode, we devise an additional *extended* mode which allows for even more flexibility, by also including linear combinations of the input parameters; we call this mode AO PGV-ELC, and formally define it as follows:

Definition 13 (AO PGV-ELC mode)**.** Given a κ-n-elements blockcipher E over some prime field $\mathbb{F}_p$, the input sizes $\kappa' \leq \kappa$

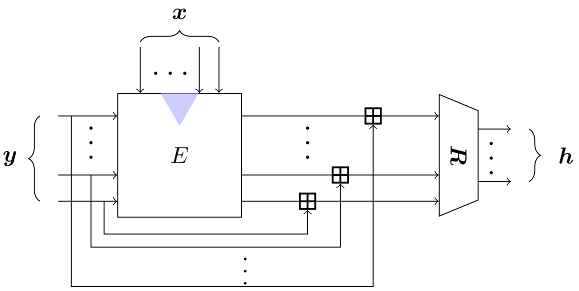

Fig. 1: Pictorial representation of the AO PGV-LC mode as by Definition 12.

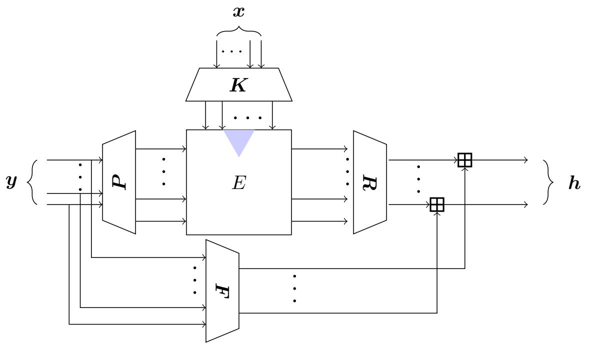

Fig. 2: Pictorial representation of the AO PGV-ELC mode as by Definition 13.

and $n' \leq n$, the output size $l \leq n'$, a left invertible key matrix $\boldsymbol{K} \in \mathbb{F}_p^{\kappa\times\kappa'}$, a left invertible plaintext matrix $\boldsymbol{P} \in \mathbb{F}_p^{n\times n'}$, a right invertible feedback matrix $\boldsymbol{F} \in \mathbb{F}_p^{l\times n'}$, and a right invertible reduction matrix $\boldsymbol{R} \in \mathbb{F}_p^{l\times n}$, the AO PGV-ELC mode of E is the compression function:

$$C_{E,V}(\boldsymbol{x},\boldsymbol{y})\colon \mathbb{F}_p^{\kappa'}\times\mathbb{F}_p^{n'}\to\mathbb{F}_p^{l} = \boldsymbol{R}E_{(\boldsymbol{Kx})}(\boldsymbol{Py})+\boldsymbol{Fy}$$

where $V = (\boldsymbol{K},\boldsymbol{P},\boldsymbol{F},\boldsymbol{R})$.

Again, the invertibility properties of the various matrices are required to guarantee the security of this construction, as we will show in Section IV. A pictorial representation of the AO PGV-ELC mode is given in Figure 2.

IV. Security Proofs

In [47], it was shown that among the 64 bit-oriented PGV-MD iterated compression modes, each denoted with $C^{(\iota)}(x,y)$, the first twelve of them, called Group-1 modes and shown in Table I, are collision and preimage resistant both when used for MD hashing and when used for 2-1 compression by replacing the role of the chaining value with a second message block.

In the design phase of the AO PGV-LC and the PGV-ELC mode, we followed the patterns that emerge from the structure of the classical Group-1 construction: first, notice how the 12 modes are pairwise symmetric, and only modes 1 and 5 are minimal w.r.t. the number of extra additions required. As we will see, an argument similar to the one given in [47] is enough to guarantee the security of the PGV-LC mode.

TABLE I: The AO equivalent of the 12 Group-1 PGV compression modes of [47]. Note that modes 5–8 are completely symmetric to modes 1–4. Similarly, mode 9 is symmetric to mode 10, and mode 11 is symmetric to mode 12.

ι	$C^{(\iota)}(\boldsymbol{x},\boldsymbol{y})$	ι	$C^{(\iota)}(\boldsymbol{x},\boldsymbol{y})$
1	$E_{\boldsymbol{x}}(\boldsymbol{y})+\boldsymbol{y}$	7	$E_{\boldsymbol{y}}(\boldsymbol{x})+\boldsymbol{x}+\boldsymbol{y}$
2	$E_{\boldsymbol{x}}(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{x}+\boldsymbol{y}$	8	$E_{\boldsymbol{y}}(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{x}$
3	$E_{\boldsymbol{x}}(\boldsymbol{y})+\boldsymbol{x}+\boldsymbol{y}$	9	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{y})+\boldsymbol{y}$
4	$E_{\boldsymbol{x}}(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{y}$	10	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{x})+\boldsymbol{x}$
5	$E_{\boldsymbol{y}}(\boldsymbol{x})+\boldsymbol{x}$	11	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{y})+\boldsymbol{x}$
6	$E_{\boldsymbol{y}}(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{x}+\boldsymbol{y}$	12	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{x})+\boldsymbol{y}$

A. *Security of AO* PGV-LC *mode*

Theorem 1 (COMP-COL resistance of AO PGV-LC mode)**.** *Given the κ-n-elements ideal AO blockcipher E over some prime field $\mathbb{F}_p$, some $l < n$, a number of queries $q < p^l$, a right invertible matrix $\boldsymbol{R} \in \mathbb{F}_p^{l\times n}$, and the $(\kappa+n)$-l-elements AO* PGV-LC *compression function $C_{E,\boldsymbol{R}}$, it holds that:*

$$\mathbf{Adv}_{C_{E,\boldsymbol{R}}}^{\text{COMP-COL}}(q) \leq \frac{q^2+q}{p^l-q}$$

Proof. Let $\mathcal{E}_q$ be the oracle implementing E and responding to at most q queries, as depicted in Algorithm 1. Let $\mathcal{A}^{\mathcal{E}_q}$ be any adversary with oracle access to $\mathcal{E}_q$. Let Col be the event that $\mathcal{A}^{\mathcal{E}_q}$ finds $\boldsymbol{x},\boldsymbol{x}' \in \mathbb{F}_p^\kappa$ and $\boldsymbol{y},\boldsymbol{y}' \in \mathbb{F}_p^n$ such that $(\boldsymbol{x},\boldsymbol{y}) \neq (\boldsymbol{x}',\boldsymbol{y}')$ and $\boldsymbol{h}=\boldsymbol{h}'$, with $\boldsymbol{h}=C_{E,\boldsymbol{R}}(\boldsymbol{x},\boldsymbol{y})$ and $\boldsymbol{h}'=C_{E,\boldsymbol{R}}(\boldsymbol{x}',\boldsymbol{y}')$. Clearly, $\Pr[\mathsf{Col}] = \mathbf{Adv}_{C_{E,\boldsymbol{R}}}^{\text{COMP-COL}}(\mathcal{A})$. Without loss of generality, we can make the following assumptions:

1) $\mathcal{A}$ makes exactly q queries to $\mathcal{E}_q$.
2) $\mathcal{A}$ keeps track of the query list $\mathcal{Q} = (Q_i)_{i\in\{1,\dots,q\}}$, where in each $Q_i = (\boldsymbol{x}_i,\boldsymbol{y}_i,\boldsymbol{c}_i,b_i)$, $\boldsymbol{x}_i \in \mathbb{F}_p^\kappa$ is the queried key, $b_i \in \{0,1\}$ is the queried selection bit, and if $b_i = 0$, then $\boldsymbol{y}_i \in \mathbb{F}_p^n$ is the queried plaintext, while $\boldsymbol{c}_i \in \mathbb{F}_p^n$ is the returned ciphertext; otherwise, $\boldsymbol{c}_i$ is the queried ciphertext and $\boldsymbol{y}_i$ the returned plaintext.
3) If $\mathcal{A}$ finds a collision, there are $Q_i, Q_j \in \mathcal{Q}$ such that $\boldsymbol{h}_i = \boldsymbol{R}(\boldsymbol{c}_i+\boldsymbol{y}_i) = \boldsymbol{h}_j = \boldsymbol{R}(\boldsymbol{c}_j+\boldsymbol{y}_j)$.

Since $\boldsymbol{R}$ is right invertible, it induces a partition of $\mathbb{F}_p^n$ into p^l equivalence classes $[\boldsymbol{v}]_{\boldsymbol{R}}$, one for each $\boldsymbol{v} \in \mathbb{F}_p^l$. We will now drop $\boldsymbol{R}$ from the subscript for ease of presentation. Clearly, $|[\boldsymbol{v}]| = p^{n-l}$. Given any $\boldsymbol{u},\boldsymbol{w} \in \mathbb{F}_p^n$, and any $\boldsymbol{v} \in \mathbb{F}_p^l$, if $\boldsymbol{u}+\boldsymbol{w} \in [\boldsymbol{v}]$ we say that $\boldsymbol{u}$ is *$\boldsymbol{w}$-$\boldsymbol{v}$-linking* (via $\boldsymbol{R}$). Note that then it is also true that $\boldsymbol{w}$ is $\boldsymbol{u}$-$\boldsymbol{v}$-linking. Let $L_{\boldsymbol{w},\boldsymbol{v}}$ be the set of all $\boldsymbol{w}$-$\boldsymbol{v}$-linking values of $\boldsymbol{u}$: since $\boldsymbol{u}$ and $\boldsymbol{w}$ come from the same vector space, and that addition is a permutation over one its arguments, we have that $|L_{\boldsymbol{w},\boldsymbol{v}}| = p^{n-l}$.

Given any queries $Q_i, Q_j \in \mathcal{Q}$, let $\mathsf{Link}_{i,j}$ be the event that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}_j$-linking Observe that $\mathsf{Link}_{i,j} = \mathsf{Link}_{j,i}$. Then:

$$\Pr[\mathsf{Col}] = \Pr[\exists i < j \leq q\colon \mathsf{Link}_{i,j}] = \Pr[\mathsf{Link}_{0,1} \vee \cdots \vee \mathsf{Link}_{q-1,q}]$$

Let's consider each combination of b and b':

- $b_i = b_j = 0$: $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are freely chosen among at least $p^\kappa - q$ possible values, while $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are freely chosen among at least $p^n - q$ possible values. $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ are then random values from sets of cardinality at least $p^n - q$.

Then, independently of how $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ were chosen, $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are also random. There are at most p^{n-l} values of $\boldsymbol{y}_i$ which are $\boldsymbol{c}_i$-$\boldsymbol{h}_j$-linking, hence:

$$\Pr[\mathsf{Col}] \leq \sum_{j=1}^{q} \sum_{i=1}^{j} \frac{p^{n-l}}{p^n - q} \leq \sum_{j=1}^{q} \sum_{i=1}^{j} \frac{1}{p^l - q} \leq \frac{q^2 + q}{p^l - q}$$

- $b_i = b_j = 1$: $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ are all freely chosen by $\mathcal{A}$, with $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ coming from sets of size at least $p^n - q$. This time, $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are random, and the same reasoning as before applies: once again, $\Pr[\mathsf{Col}] \leq (q^2 + q)/(p^l - q)$.
- $b_i = 0 = 1 - b_j$: in this case, $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{y}_i$ and $\boldsymbol{c}_j$ are freely chosen by $\mathcal{A}$. $\boldsymbol{c}_i$ and $\boldsymbol{y}_j$ are random, independently of which among $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ was found earlier, the probability that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}_j$ linking is at most $\Pr[\mathsf{Col}] \leq (q^2 + q)/(p^l - q)$.
- $b_j = 0 = 1 - b_i$: similar as before.

Since all the probabilities given above depend only on the number of queries made by $\mathcal{A}$, and not on its behaviour, the claim follows. □

Theorem 2 (COMP-PRE resistance of AO PGV-LC mode)**.** *Given the κ-n-elements ideal AO blockcipher E over some prime field $\mathbb{F}_p$, some $l < n$, a number of queries $q < p^l$, a right-invertible matrix $\boldsymbol{R}$, and the $(\kappa + n)$-l-elements AO* PGV-LC *compression function $C_{E,\boldsymbol{R}}$, it holds that:*

$$\mathbf{Adv}^{\text{COMP-PRE}}_{C_{E,\boldsymbol{R}}}(q) \leq \frac{q}{p^l - q}$$

Proof. We start from the setup that we developed in the proof of Theorem 1. Given some random $\boldsymbol{h} \in \mathbb{F}_p^l$, let Pre be the event that $\mathcal{A}^{\mathcal{E}_q}$ finds some $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}_p^\kappa \times \mathbb{F}_p^n$ such that $C_{E,\boldsymbol{R}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{h}$. Clearly, $\Pr[\mathsf{Pre}] = \mathbf{Adv}^{\text{COMP-PRE}}_{C_{E,\boldsymbol{R}}}(\mathcal{A})$. Now let Link_i be the event that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}$-linking, then $\Pr[\mathsf{Pre}] = \Pr[\exists i \leq q\colon \mathsf{Link}_i]$. We have two cases to consider:

- $b_i = 0$: $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are chosen arbitrarily, and $\boldsymbol{c}_i$ is a random element from a set of size at least $p^n - q$, and there are at most p^{n-l} values of $\boldsymbol{y}_i$ that are $\boldsymbol{c}_i$-$\boldsymbol{h}$-linking. Hence, $\Pr[\mathsf{Pre}] \leq \sum_{i=1}^{q} \frac{1}{p^l - q} \leq \frac{q}{p^l - q}$.
- $b_i = 1$: $\boldsymbol{x}_i$ and $\boldsymbol{c}_i$ are chosen arbitrarily, and $\boldsymbol{y}_i$ is random, as before we can then conclude that $\Pr[\mathsf{Pre}] \leq q/(p^l - q)$.

Since the probability of finding a preimage does not depend on the behaviour of $\mathcal{A}$, the claim follows. □

B. Security of AO PGV-ELC *mode*

The main difference between the AO PGV-LC and PGV-ELC modes is that the latter allows for input sizes to the compression function which do not necessarily match the plaintext or key sizes of the underlying blockcipher. Intuitively, this additional flexibility should not impact the security, but one must be careful when considering that the input entropy pool is reduced, as now part of the plaintext/ciphertext and key space might be left unused.

Theorem 3 (COMP-COL resistance of AO PGV-ELC mode)**.** *Given the κ-n-elements ideal AO blockcipher E over some prime field $\mathbb{F}_p$, the $(\kappa' + n')$-l-elements AO* PGV-ELC *compression function $C_{E,V}$, where κ', n' and $V = (\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{F}, \boldsymbol{R})$ are as in Definition 13, and a number of queries $q < p^l$, it holds that:*

$$\mathbf{Adv}^{\text{COMP-COL}}_{C_{E,V}}(q) \leq \frac{q^2 + q}{p^l - q}$$

Sketch. The proof builds on the setup used in Theorem 1, generalizing some concepts to account for the extended setting. See Appendix A for the full proof. □

Now that we have proven collision resistance of our construction, we turn to preimage resistance:

Theorem 4 (COMP-PRE resistance of AO PGV-ELC mode)**.** *Given the κ-n-elements ideal AO blockcipher E over some prime field $\mathbb{F}_p$, some $l < n$, a number of queries $q < p^l$, and the $(\kappa' + n')$-l-elements AO* PGV-ELC *compression function $C_{E,V}$, where κ', n' and $V = (\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{F}, \boldsymbol{R})$ are as in Definition 13, it holds that:*

$$\mathbf{Adv}^{\text{COMP-PRE}}_{C_{E,V}}(q) \leq \frac{q}{p^l - q}$$

Sketch. The proof is similar to the case of Theorem 2, exploiting the additional insights detailed in the proof of Theorem 3. See Appendix A for the full proof. □

C. Security of AO t-ary Merkle Tree

We can now turn to consider collision resistance for the Merkle tree hashing: the classical result over bit-strings generalizes trivially to AO constructions.

Theorem 5 (HASH-COL resistance of AO Merkle tree)**.** *Given a tn-n elements compression function family C over a prime field $\mathbb{F}_p$, and a number of queries $q < p^n$, it holds that:*

$$\mathbf{Adv}^{\text{HASH-COL}}_{H_C}(q) \leq \mathbf{Adv}^{\text{COMP-COL}}_{C}(q) + \mathbf{Adv}^{\text{COMP-PRE}}_{C}(q)$$

where H_C is the Merkle tree mode of hashing family over C.

Sketch. The proof generalizes the standard ideas used in the well-known case of binary Merkle trees over bit-oriented compression functions. See Appendix A for the full proof. □

The last thing we need to prove, which again is a relatively straightforward adaptation of a classical result, is opening resistance of the AO Merkle tree. In this setting, we are given a t-ary Merkle tree $\mathcal{T}_{C,M}$ over some tn-n elements compression function C and some message $M \in (\mathbb{F}_p^n)^{t^h}$ (i.e. we assume M to fit exactly in the tree). Only C and the root of the node, $\boldsymbol{\nu}_0 = H(M)$, are known to the verifier $\mathcal{V}$. Let $t' = t - 1$; in order to check membership of some leaf $\boldsymbol{\nu}_i$ in $\mathcal{T}_{C,M}$, the generator $\mathcal{G}$ sends to $\mathcal{V}$ the opening proof $\pi = (i, \boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{ht'})$, where we expect $\boldsymbol{x}_0$ to be $\boldsymbol{\nu}_i$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{ht'}$ to be the nodes in the co-path from $\boldsymbol{\nu}_i$ to $\boldsymbol{\nu}_0$. Then, $\mathcal{V}$ takes the base-t digit expansion $(d_{h-1}, \ldots, d_0)$ of the index i and collects consecutive elements of the co-path in groups of t' units: each digit will fix the position of the chaining value c_j, so that $\boldsymbol{c}_0 = 0$ and $\forall j < h$:

$$\boldsymbol{c}_{j+1} = C\big(\boldsymbol{x}_{t'j+1}, \dots, \boldsymbol{x}_{t'j+d_j-1}, \boldsymbol{c}_j, \boldsymbol{x}_{t'j+d_j}, \dots, \boldsymbol{x}_{t'j+t'}\big)$$

Finally, $\mathcal{V}$ compares $\boldsymbol{c}_h$ with $\boldsymbol{\nu}_0$: if they are equal, it accepts, otherwise it rejects.

Theorem 6 (OPENING resistance of AO Merkle tree)**.** *Given a tn-n elements compression function family C over some prime field $\mathbb{F}_p$, and a number of queries $q < p^n$, it holds that:*

$$\mathbf{Adv}^{\text{OPENING}}_{H_C}(q) \le \mathbf{Adv}^{\text{COMP-COL}}_{C}(2q)$$

where H_C is the Merkle tree mode of hashing family over C.

Proof. Consider the t-ary Merkle tree $\mathcal{T}_{C,M}$ over a message $M \in \left(\mathbb{F}_p^n\right)^{t^h}$, and let $t' = t - 1$. Now, let $\mathcal{C}$ be the oracle implementing C, and let $\mathcal{A}$ be an adversary making q queries to $\mathcal{C}$ that can forge a proof $\tilde{\pi} = (i, \tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_1, \dots, \tilde{\boldsymbol{x}}_{ht'})$ with advantage $\mathbf{Adv}^{\text{OPENING}}_{H_C}(\mathcal{A})$. We will now build an adversary $\mathcal{B}$ which finds a collision in $C^{(\iota)}$ as follows: first, $\mathcal{B}$ runs $\pi = \mathcal{G}(M, i)$ and $\tilde{\pi} = \mathcal{A}(M, i)$. Then, it computes the correct chaining values $\boldsymbol{c}_0$ through $\boldsymbol{c}_h$, and the forged chaining values $\tilde{\boldsymbol{c}}_0$ through $\tilde{\boldsymbol{c}}_h$: by completeness of $(\mathcal{G}, \mathcal{V})$, we have that $\boldsymbol{c}_h = \boldsymbol{\nu}_0$. Now $\mathcal{B}$ compares $\boldsymbol{c}_h$ with $\tilde{\boldsymbol{c}}_h$: if the two of them are different, it halts rejecting as $\mathcal{A}$ did not actually find a collision. Otherwise, it computes the base-t digit expansion $(d_{h-1}, \dots, d_0)$ of i and starts matching, for $j \in \{h-1, \dots, 0\}$, $\boldsymbol{c}_j$ with $\tilde{\boldsymbol{c}}_j$ and $\boldsymbol{x}_{jt'+1}, \dots, \boldsymbol{x}_{jt'+t'}$ with $\tilde{\boldsymbol{x}}_{jt'+1}, \dots, \tilde{\boldsymbol{x}}_{jt'+t'}$: if the match is only partial, then the two vectors:

$$\boldsymbol{m}_j = \begin{pmatrix}\boldsymbol{x}_{t'j+1} & \dots & \boldsymbol{x}_{t'j+d_j-1} & \boldsymbol{c}_j & \boldsymbol{x}_{t'j+d_j} & \dots & \boldsymbol{x}_{t'j+t'}\end{pmatrix}^{\mathsf{T}}$$
$$\tilde{\boldsymbol{m}}_j = \begin{pmatrix}\tilde{\boldsymbol{x}}_{t'j+1} & \dots & \tilde{\boldsymbol{x}}_{t'j+d_j-1} & \tilde{\boldsymbol{c}}_j & \tilde{\boldsymbol{x}}_{t'j+d_j} & \dots & \tilde{\boldsymbol{x}}_{t'j+t'}\end{pmatrix}^{\mathsf{T}}$$

form a collision, since $\boldsymbol{c}_{j+1} = C(\boldsymbol{m}_j) = \tilde{\boldsymbol{c}}_{j+1} = C(\tilde{\boldsymbol{m}}_j)$, hence $\mathcal{B}$ will return the pair $(\boldsymbol{m}, \tilde{\boldsymbol{m}})$, and it will halt accepting. Finally, if all the matches up to $j = 0$ are exact, then it must be the case that $\pi = \tilde{\pi}$, therefore the forged proof is actually a valid proof, so $\mathcal{B}$ will halt rejecting. We can then conclude that $\mathcal{B}$ finds a valid collision for C whenever $\mathcal{A}$ finds a valid opening proof forgery for H_C: assuming w.l.o.g. that $\mathcal{A}$ had to perform at least the h oracle queries required to compute the root of the tree (i.e. $h < q$), and that $\mathcal{B}$ needs to call $\mathcal{G}$ in order to compute π, the claim follows. □

V. Implementations and Experiments

In order to assess the efficiency of the PGV-ELC mode, among the many available arithmetization-oriented constructions, we decided to select the HADES-MiMC design [51]: firstly, it is a blockcipher design, so it can be instantiated over the PGV-ELC mode; secondly, it has undergone (and resisted to) a good amount of cryptanalysis; thirdly, the Sponge hash function derived from HADES-MiMC, i.e. POSEIDON [28], is quite popular in the industry, and it is deployed in several real-world systems [14], [37], where even small improvements requiring relatively minor changes can be meaningful; finally, it is well-defined for any arbitrary block size, allowing us a good level of flexibility for our experiments (however, see Remark 2).

An optimized version of POSEIDON, dubbed POSEIDON2, was recently proposed [30]: the two main differences with POSEIDON are the usage of an efficiently computable MDS matrix for the linear layer, and support of the Trunc compression mode. In our experiments, we will compare both the Sponge-based POSEIDON and the Trunc-based POSEIDON2 with the corresponding PGV-ELC instantiation of HADES-MiMC.

Definition 14 (POSEIDON-DM)**.** Let $E\colon \mathbb{F}_p^n \times \mathbb{F}_p^n \to \mathbb{F}_p^n$ be the HADES-MiMC blockcipher as defined in [51], with the scheduling function and the linear layer both instantiated by the $n \times n$ Hilbert matrix. Additionally, given some $l, m < n$, let $\boldsymbol{K} = \boldsymbol{P} = \boldsymbol{I}^{n\times m}$, $\boldsymbol{F} = \boldsymbol{I}^{l\times m}$, and $\boldsymbol{R} = \boldsymbol{I}^{l\times n}$. Then, POSEIDON-DM is the compression function:

$$C(\boldsymbol{x}, \boldsymbol{y})\colon \mathbb{F}_p^m \times \mathbb{F}_p^m \to \mathbb{F}_p^l = \boldsymbol{R} \cdot E_{\boldsymbol{Ky}}(\boldsymbol{Px}) + \boldsymbol{Fy}$$

Definition 15 (POSEIDON2-DM)**.** POSEIDON2-DM is the compression function POSEIDON-DM, where the full-rounds and partial-rounds linear layers are instantiated respectively by the matrices $\boldsymbol{M}_{\mathcal{E}}$ and $\boldsymbol{M}_{\mathcal{I}}$ described in [30], and the key scheduling function is instantiated by the matrices:

$$\boldsymbol{M}_{\mathcal{K},\mathbf{2}} = \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} \qquad \boldsymbol{M}_{\mathcal{K},\mathbf{4}} = \begin{bmatrix}2 & 3 & 1 & 1\\ 1 & 2 & 3 & 1\\ 3 & 1 & 2 & 1\\ 1 & 1 & 1 & 2\end{bmatrix}$$

when $n = 2$ or $n = 4$, and by the matrix $\boldsymbol{M}_{\mathcal{E}}$ in all other cases.

The matrices $\boldsymbol{M}_{\mathcal{K},\mathbf{2}}$ and $\boldsymbol{M}_{\mathcal{K},\mathbf{4}}$ that we use for the key scheduling in POSEIDON2-DM are both Maximum Distance Separable (MDS) [78], [79] matrices resistant to subspace trails attacks as defined in [80], and are hence suitable to be used for scheduling.

Remark 2. When the block size is of just one element, the SPN structure of the HADES-MiMC design disappears, degenerating into the MiMC construction [26]. For this reason, in our experiments the 2:1 compression function is instantiated over the 2-elements HADES-MiMC cipher, so that the matrices $\boldsymbol{K}$ and $\boldsymbol{P}$ are rectangular. Furthermore, n:m compression functions are not restricted to be used over n-ary Merkle trees: indeed, over relatively smaller prime fields, in order to achieve a target security level for collision resistance, say 128-bits, one might have to use a 4:2 ($p \approx 2^{128}$) or an 8:4 ($p \approx 2^{64}$) compression function within a binary tree: in these two cases respectively, for either FIL Sponge, Trunc or PGV-ELC, the complexity is basically equivalent to performing 4:1 or 8:1 compression.

Experimental Setup.: All of our benchmarks were run on a system with an Intel® Core™ i9–13900KF @6.0GHz CPU equipped with 32 GB of DDR5–5200 RAM, running a Clear Linux OS 40630 instance. We wish to point out that the tested CPU is equipped with 8 ‘performance’ cores, with hyper-threading, plus 16 'efficiency' cores: this results in an unsteady behaviour when running parallel experiments due to the different types of threads. Experiments in the native settings were run with our own C++ implementation of the various primitives and Merkle trees, using either the `NTL`[1] library, the `libff`[2] library, or our own custom library (dubbed `libarith`) as backends for the finite field arithmetic. For the ZK-SNARK comparisons, we implemented the arithmetic circuits of the target primitives in the `libsnark`[3] library, which is based on the Groth16 [6] ZK-SNARK framework, and uses R1CS arithmetization. All code was compiled with the Intel® oneAPI DPC++ Compiler 2024.0.2 with compiler flags `-std=c++17 -O2 -march=native` for serial code, and all previous flags plus the `-qopenmp` flag for parallel code. As our main objective is to compare the Sponge and Trunc modes with our new modes, in order to obtain more meaningful results we ran our experiments over three different prime fields: respectively the scalar fields of the elliptic curves BLS12-381 [81], [82], BN-254 [83], [84], and Edwards-180 [85], as reported in Table VII (in appendix A).

A. Optimized R1CS for t-ary Merkle Tree

R1CS systems constrain the computation by means of a system of bilinear equations of the kind $(\boldsymbol{Ax}) \odot (\boldsymbol{Bx}) = \boldsymbol{Cx}$, where $\odot$ denotes the Hadamard (i.e. element-wise) product. It is well known how to build a R1CS system for *binary* Merkle trees[4]; however, the only public implementation that we found which also offers wider arities [86] only offers hardcoded circuits for arity $t \in \{2, 4, 8\}$. While writing our own R1CS for an arbitrary t, we found that a small change in the classical opening proof protocol (described in Section IV-C) allows for a more compact R1CS system.

For a binary tree, given the opening proof $\pi = (i, \boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_h)$, where all vectors are over $\mathbb{F}_p^n$, the prover itself will compute the chaining values $(\boldsymbol{c}_0, \ldots, \boldsymbol{c}_h)$: in order to guarantee that the order of the inputs is preserved and that the output values are correct, we introduce fresh variables $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{h-1}$, and use the binary expansion $(d_{h-1}, \ldots, d_0)$ of i as selector bits. We then enforce, for $0 \leq j < h$:

$$\begin{cases} d_j \cdot (1 - d_j) = 0 \\ d_j \cdot (\boldsymbol{c}_j - \boldsymbol{x}_{j+1}) = \boldsymbol{c}_j - \boldsymbol{y}_j \\ \boldsymbol{c}_{j+1} = C(\boldsymbol{y}_j, \boldsymbol{c}_j + \boldsymbol{x}_{j+1} - \boldsymbol{y}_j) \end{cases}$$

This system requires $h(1 + n + R_C)$ constraints, where R_C is the number of constraints required to instantiate C.

TABLE II: Number of R1CS constraints in the baseline and optimized MT circuits over POSEIDON-DM, for trees with 2^{24} leaves, each containing n field elements.

		MT Arity		
	n	4	8	16
Baseline	1	3000	2552	2754
	2	3696	3520	4182
	4	5124	5408	7056
	8	7908	9208	12768
Optimized	1	2916	2392	2484
	2	3540	3248	3732
	4	4824	4912	6246
	8	7320	8264	11238
Improvement	1	2.88%	6.69%	10.9%
	2	4.41%	8.37%	12.1%
	4	6.22%	10.1%	13.0%
	8	8.03%	11.4%	13.6%

One possible way to generalize to any arity $t \geq 2$, similar to the one used in [86], is to consider the authentication path $\pi = (i, \boldsymbol{x}_0, \boldsymbol{x}_{0,1}, \ldots, \boldsymbol{x}_{h-1,t'})$, where $t' = t - 1$, together with the base-t expansion $(d_{h-1}, \ldots, d_0)$ of the index i. Now, $\boldsymbol{c}_{j+1} = C(\boldsymbol{y}_{j,1}, \ldots, \boldsymbol{y}_{j,t})$ where, depending on d_j, $\boldsymbol{y}_{j,1}$ could be either $\boldsymbol{c}_j$ or $\boldsymbol{x}_{j,1}$, $\boldsymbol{y}_{j,t}$ could be either $\boldsymbol{c}_j$ or $\boldsymbol{x}_{j,t-1}$, and any other $\boldsymbol{y}_{j,k}$ could be either $\boldsymbol{c}_j$, $\boldsymbol{x}_{j,k-1}$ or $\boldsymbol{x}_{j,k}$. Let $b = \lceil \log_2(t) \rceil$, and consider the binary expansion $(d_{j,b}, \ldots, d_{j,1})$ of d_j: we can compute all possible combinations of these binary values and store them in the selector variables $s_{j,1}, \ldots, s_{j,t}$: if we do it in a tree-like fashion, we need $2^{b+1} - 4 = 2t - 4$ multiplications to do so. Hence, we can set up a constraint system equivalent to the following, for $0 \leq j < h$:

$$\begin{cases} \forall 1 \leq k \leq b \colon d_{j,k} \cdot (1 - d_{j,k}) = 0 \\ \prod_{k=1}^{b} (1 - d_{j,k}) = s_{j,1} \\ \cdots \\ \prod_{k=1}^{b} d_{j,k} = s_{j,t} \\ \forall k \colon (s_{j,k} \cdot \boldsymbol{c}_j) + (s_{j,k-1} \cdot \boldsymbol{x}_{jt'+k-1}) + (\tilde{s}_{j,k} \cdot \boldsymbol{x}_{jt'+k}) = \boldsymbol{y}_{j,k} \\ \boldsymbol{c}_{j+1} = C(\boldsymbol{y}_{j,1}, \ldots, \boldsymbol{y}_{j,t}) \end{cases}$$

Where $\tilde{s}_{i,j}$ is a shorthand for $(1 - s_{i,j})$. This constraint system requires a total of $h(b + 2t - 4 + n(4 + 3(t-2)) + R_C)$ constraints.

Now, consider the modified opening proof where the prover sends, together with all the others, also the node that the verifier is able to compute by itself. With this slight modification, we can then introduce as before the selector variables $s_{0,1}, \ldots, s_{h-1,t}$, and enforce, for each $0 \leq j < h$:

$$\begin{cases} \forall 1 \leq k \leq t \colon s_{j,k} \cdot (1 - s_{j,k}) = 0 \\ 1 \cdot \sum_{k=1}^{t} s_{j,k} = 1 \\ \forall 1 \leq k \leq t \colon s_{j,k} \cdot (\boldsymbol{c}_j - \boldsymbol{x}_{j,k}) = \boldsymbol{c}_j - \boldsymbol{y}_{j,k} \\ \boldsymbol{c}_{j+1} = C(\boldsymbol{y}_{j,1}, \ldots, \boldsymbol{y}_{j,t}) \end{cases}$$

The optimized constraint system then requires $h(t + 1 + tn + R_C)$ constraints. The relative improvement we can get by using the optimized circuit is therefore:

$$\frac{\lceil \log_2(t) \rceil + 2t - 4 + n(4 + 3(t-2)) + R_C}{t + 1 + tn + R_C}$$

which is independent of the tree height, and results in higher improvements the smaller the compression primitive constraint system is. In Table II, we show the concrete improvement over

[1] https://libntl.org

[2] https://github.com/scipr-lab/libff

[3] https://github.com/scipr-lab/libsnark

[4] See for example: https://github.com/arkworks-rs/r1cs-tutorial.

TABLE III: Number of R1CS constraints required to represent the target compression functions.

	Compression Rate		
Primitive	2:1	4:1	8:1
POSEIDON	237	288	384
POSEIDON-DM	213	213	261
POSEIDON2-Trunc	213	264	360
POSEIDON2-DM	213	213	261
Constraint Reduction			
POSEIDON-DM	-11%	-35%	-47%
POSEIDON2-DM	0%	-24%	-38%

TABLE IV: Time to generate a Merkle tree opening proof in the Groth16 framework (`libsnark`, $|M| = 2^{30}$).

		MT Arity		
Primitive	Field	2	4	8
POSEIDON	BLS-381	1.35 s	0.846 s	0.761 s
	BN-254	0.837 s	0.525 s	0.470 s
	Ed-180	0.440 s	0.281 s	0.253 s
POSEIDON-DM	BLS-381	1.24 s	0.671 s	0.538 s
	BN-254	0.767 s	0.413 s	0.333 s
	Ed-180	0.398 s	0.215 s	0.178 s
POSEIDON2	BLS-381	1.29 s	0.807 s	0.756 s
	BN-254	0.779 s	0.488 s	0.453 s
	Ed-180	0.403 s	0.257 s	0.240 s
POSEIDON2-DM	BLS-381	1.30 s	0.692 s	0.554 s
	BN-254	0.779 s	0.413 s	0.333 s
	Ed-180	0.403 s	0.216 s	0.177 s
Average speed-up				
POSEIDON-DM		$+10\%$	$+28\%$	$+42\%$
POSEIDON2-DM		0%	$+18\%$	$+36\%$

Merkle Trees of different arities and node sizes when C is instantiated by either POSEIDON-DM or POSEIDON2-DM.

B. ZK-SNARK Performance

The main bottleneck of ZK-SNARK frameworks usually lies in the time needed to generate the proof, hence we target proof generation time as our efficiency metric. In turn, the complexity of building a proof varies on the framework: for R1CS-based ZK-SNARK systems, it primarily depends on the multiplicate complexity (i.e. number of equations) in the constraint system itself, and secondarily on its sparsity: a lower number of constraints is normally directly related to an improvement in the resulting proof generation time.

Since the only difference between the POSEIDON and POSEIDON2 constructions is the affine layer, they both require an equal number of R1CS constraints for the same state size, and the same holds for POSEIDON-DM and POSEIDON2-DM, as can be seen in Table III. However, as the Trunc mode allows POSEIDON2 to save one state element compared to Sponge, similarly the PGV-ELC mode allows us to halve the number of state elements, resulting in a significant reduction of R1CS constraints.

In Table IV and Figure 3 we see how the theoretical numbers reflect also in the empirical experiments: as expected, there is but negligible difference between POSEIDON-DM and POSEIDON2-DM. More interestingly, we can see how increasing the arity improves proof generation time across the board: for example, generating a SNARK for an opening proof over octonary trees with either POSEIDON-DM or POSEIDON2-DM is about 2.5× faster than with either POSEIDON and POSEIDON2 over binary trees.

C. Native Performance

TABLE V: Time to compute a single primitive call. Note that 2:1 compression is a special case, see Remark 2.

NTL				
		Compression rate		
Primitive	Field	2:1*	4:1	8:1
POSEIDON	BLS-381	47.4 µs	112 µs	337 µs
	BN-254	48.2 µs	114 µs	344 µs
	Ed-180	39.9 µs	95.9 µs	299 µs
POSEIDON-DM	BLS-381	40.4 µs	40.8 µs	133 µs
	BN-254	41.5 µs	41.5 µs	135 µs
	Ed-180	35.4 µs	35.1 µs	118 µs
POSEIDON2-Trunc	BLS-381	13.5 µs	26.6 µs	46.5 µs
	BN-254	13.5 µs	26.4 µs	45.3 µs
	Ed-180	11.5 µs	30.6 µs	43.3 µs
POSEIDON2-DM	BLS-381	15.0 µs	15.0 µs	28.2 µs
	BN-254	15.5 µs	15.3 µs	28.3 µs
	Ed-180	13.3 µs	13.1 µs	26.0 µs
libff				
POSEIDON	BLS-381	18.0 µs	47.2 µs	157 µs
	BN-254	18.1 µs	45.0 µs	142 µs
	Ed-180	17.0 µs	41.8 µs	122 µs
POSEIDON-DM	BLS-381	14.9 µs	15.7 µs	59.4 µs
	BN-254	15.2 µs	15.6 µs	56.2 µs
	Ed-180	14.4 µs	14.2 µs	49.0 µs
POSEIDON2-Trunc	BLS-381	5.44 µs	13.9 µs	23.8 µs
	BN-254	5.47 µs	13.9 µs	23.6 µs
	Ed-180	5.35 µs	12.7 µs	21.8 µs
POSEIDON2-DM	BLS-381	6.39 µs	6.44 µs	15.4 µs
	BN-254	6.44 µs	6.56 µs	15.6 µs
	Ed-180	6.51 µs	6.51 µs	15.1 µs
libarith				
POSEIDON	BLS-381	11.6 µs	26.3 µs	74.1 µs
	BN-254	11.8 µs	26.4 µs	76.2 µs
	Ed-180	8.48 µs	19.2 µs	55.1 µs
POSEIDON-DM	BLS-381	9.89 µs	10.0 µs	32.7 µs
	BN-254	10.4 µs	10.4 µs	33.0 µs
	Ed-180	7.46 µs	7.55 µs	24.7 µs
POSEIDON2-Trunc	BLS-381	3.30 µs	7.32 µs	12.6 µs
	BN-254	3.24 µs	7.11 µs	12.1 µs
	Ed-180	2.51 µs	5.48 µs	8.97 µs
POSEIDON2-DM	BLS-381	3.51 µs	3.50 µs	7.88 µs
	BN-254	3.47 µs	3.48 µs	7.75 µs
	Ed-180	2.67 µs	2.69 µs	5.85 µs
Average speed-up				
NTL	POSEIDON-DM	1.17×*	2.80×	2.51×
	POSEIDON2-DM	0.86×*	1.82×	1.56×
libff	POSEIDON-DM	1.17×*	2.87×	2.57×
	POSEIDON2-DM	0.85×*	2.20×	1.57×
libarith	POSEIDON-DM	1.15×*	2.27×	2.27×
	POSEIDON2-DM	0.94×*	2.06×	1.57×

When comparing native execution times of different modes of operation, one has to be mindful of several 'implementation details': in order to try and alleviate implementation-specific differences as much as possible, in addition to using different prime fields, we also tested different arithmetic backends, and their comparison for basic arithmetic is given in Appendix A.

Then, in Table V we can see the performance of a single execution of the target primitives: observe how, due to the need of input-expansion to exploit the benefits of the partial-SPN structure, there is basically no difference between the 2:1 and the 4:1 variants of Poseidon-DM, and similarly for Poseidon2-DM. As a result, in the case of 2:1 compression, Poseidon-DM is about 15% faster than (Sponge-based) Poseidon[5], while Poseidon2-DM is between 5–15% slower than Poseidon2-Trunc. On the other hand, for all other cases, using PGV-ELC brings great efficiency improvements, with Poseidon-DM being up to three times faster than Sponge Poseidon, and Poseidon2-DM being up to two times faster than Poseidon2-Trunc.

In Figure 4 we compared the behaviour of the target primitives when embedded inside a Merkle tree to build it given an input message. The single results follow the trend expected from Table V, but once again we want to highlight the huge improvements that can be obtained by using trees with wide arity: interestingly, the best choice of arity for native computations does not necessarily match the one for ZK-SNARK generation; for example, for all target primitives but Poseidon2-Trunc, quaternary trees perform better than octonary ones, and for Poseidon octonary trees are actually the slowest. In particular, quaternary Poseidon2-DM is approximately $2.5\times$ faster than binary Poseidon2, and almost $9\times$ faster than binary Poseidon.

Finally, in Appendix A we report the behaviour of the target functions with respect to CPU parallelization: in short, as one would expect, slower primitives benefit more from parallelization as they are more computationally intensive, while being only marginally different in memory intensity.

Acknowledgments

Stefano Trevisani was supported in full and Elena Andreeva was supported in part by the Austrian Science Fund (FWF) SpyCoDe grant with number 10.55776/F8507-N.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: https://doi.org/10.1137/0218012

[2] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems," *J. ACM*, vol. 38, no. 3, pp. 690–728, jul 1991. [Online]. Available: https://doi.org/10.1145/116825.116852

[3] J. Groth, "Short non-interactive zero-knowledge proofs," in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, vol. 6477. Springer, 2010, pp. 341–358. [Online]. Available: https://www.iacr.org/archive/asiacrypt2010/6477343/6477343.pdf

[4] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," Cryptology ePrint Archive, Paper 2013/279, 2013, https://eprint.iacr.org/2013/279. [Online]. Available: https://eprint.iacr.org/2013/279

[5] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," Cryptology ePrint Archive, Paper 2013/879, 2013, https://eprint.iacr.org/2013/879. [Online]. Available: https://eprint.iacr.org/2013/879

[6] J. Groth, "On the size of pairing-based non-interactive arguments," Cryptology ePrint Archive, Paper 2016/260, 2016, https://eprint.iacr.org/2016/260. [Online]. Available: https://eprint.iacr.org/2016/260

[7] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019, https://eprint.iacr.org/2019/953. [Online]. Available: https://eprint.iacr.org/2019/953

[8] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," Cryptology ePrint Archive, Paper 2019/550, 2019, https://eprint.iacr.org/2019/550. [Online]. Available: https://eprint.iacr.org/2019/550

[9] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," Cryptology ePrint Archive, Paper 2012/215, 2012, https://eprint.iacr.org/2012/215. [Online]. Available: https://eprint.iacr.org/2012/215

[10] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018, https://eprint.iacr.org/2018/046. [Online]. Available: https://eprint.iacr.org/2018/046

[11] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 459–474.

[12] N. van Saberhagen, "Cryptonote v 2.0," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:2711472

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," Cryptology ePrint Archive, Paper 2017/1066, 2017, https://eprint.iacr.org/2017/1066. [Online]. Available: https://eprint.iacr.org/2017/1066

[14] Y. Psaras and D. Dias, "The interplanetary file system and the filecoin network," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 80–80.

[15] T. Liu, X. Xie, and Y. Zhang, "zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2968–2985. [Online]. Available: https://doi.org/10.1145/3460120.3485379

[16] A. Naveh and E. Tromer, "Photoproof: Cryptographic image authentication for any set of permissible transformations," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 255–271.

[17] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *Applied Cryptography and Network Security*, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 467–482.

[18] T. Ventali, "Awesome zero knowledge: A curated list of awesome zk resources, libraries, tools and more," GitHub Repository, 2024, https://github.com/ventali/awesome-zk. [Online]. Available: https://github.com/ventali/awesome-zk

[19] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 738–768.

[20] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," Cryptology ePrint Archive, Paper 2019/1076, 2019, https://eprint.iacr.org/2019/1076. [Online]. Available: https://eprint.iacr.org/2019/1076

[21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, "Halo infinite: Proof-carrying data from additive polynomial commitments," in *Advances in Cryptology - CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 649–680. [Online]. Available: https://doi.org/10.1007/978-3-030-84242-0_23

[22] Q. H. Dang, *Secure Hash Standard*. National Institute of Standards and Technology, Jul 2015. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.180-4

[5] We could have obtained a much higher speed-up by using the efficient key-scheduling matrix of Poseidon2-DM also in Poseidon-DM: however, we considered that such comparison would not have been fair, as matrix optimization is the whole point of Poseidon2 in the first place.

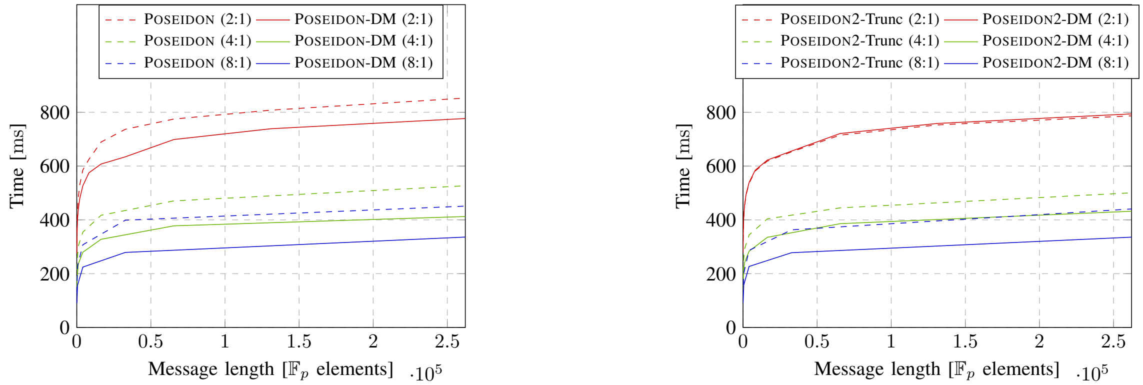

Fig. 3: Time to generate a Merkle tree opening proof over the BLS-381 field with the target primitives and arities (`libsnark`).

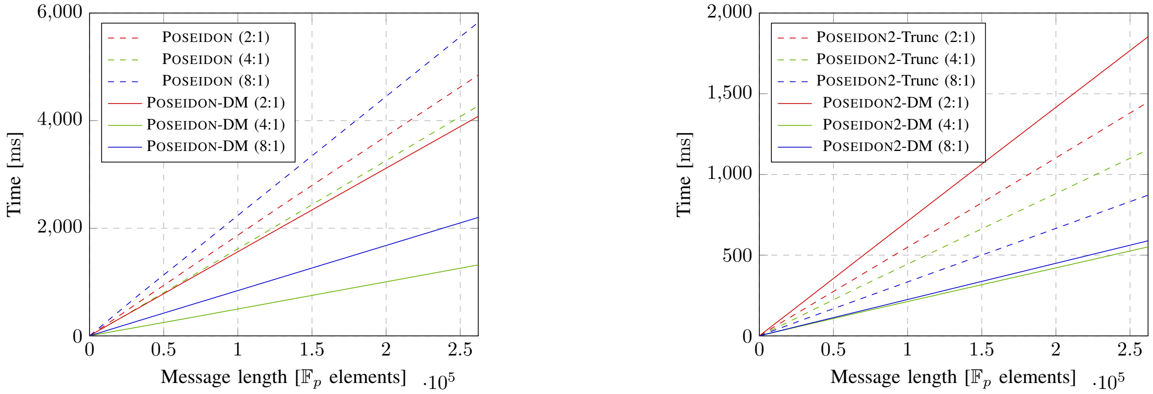

Fig. 4: Time to build a Merkle tree of different arities over the BLS-381 field with the target primitives (`libff`).

[23] L. Grassi, D. Khovratovich, R. Lüftenegger, C. Rechberger, M. Schofnegger, and R. Walch, "Reinforced concrete: A fast hash function for verifiable computation," Cryptology ePrint Archive, Paper 2021/1038, 2021, https://eprint.iacr.org/2021/1038. [Online]. Available: https://eprint.iacr.org/2021/1038

[24] C. Bouvier, P. Briaud, P. Chaidos, L. Perrin, R. Salen, V. Velichkov, and D. Willems, "New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and jive compression mode," Cryptology ePrint Archive, Paper 2022/840, 2022, https://eprint.iacr.org/2022/840. [Online]. Available: https://eprint.iacr.org/2022/840

[25] L. Grassi, Y. Hao, C. Rechberger, M. Schofnegger, R. Walch, and Q. Wang, "Horst meets fluid-spn: Griffin for zero-knowledge applications," Cryptology ePrint Archive, Paper 2022/403, 2022, https://eprint.iacr.org/2022/403. [Online]. Available: https://eprint.iacr.org/2022/403

[26] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *Advances in Cryptology – ASIACRYPT 2016*, J. H. Cheon and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 191–219.

[27] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, "Design of symmetric-key primitives for advanced cryptographic protocols," Cryptology ePrint Archive, Paper 2019/426, 2019, https://eprint.iacr.org/2019/426. [Online]. Available: https://eprint.iacr.org/2019/426

[28] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," Cryptology ePrint Archive, Paper 2019/458, 2019, https://eprint.iacr.org/2019/458. [Online]. Available: https://eprint.iacr.org/2019/458

[29] A. Szepieniec, "On the use of the legendre symbol in symmetric cipher design," Cryptology ePrint Archive, Paper 2021/984, 2021, https://eprint.iacr.org/2021/984. [Online]. Available: https://eprint.iacr.org/2021/984

[30] L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A faster version of the poseidon hash function," Cryptology ePrint Archive, Paper 2023/323, 2023, https://eprint.iacr.org/2023/323. [Online]. Available: https://eprint.iacr.org/2023/323

[31] A. Roy, M. J. Steiner, and S. Trevisani, "Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems," 2023.

[32] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger, "Feistel structures for mpc, and more," in *Computer Security – ESORICS 2019*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds. Cham: Springer International Publishing, 2019, pp. 151–171.

[33] A. Roy and M. Steiner, "Generalized triangular dynamical system: An algebraic system for constructing cryptographic permutations over finite fields," 2022, https://arxiv.org/abs/2204.01802. [Online]. Available: https://arxiv.org/abs/2204.01802

[34] A. Szepieniec, A. Lemmens, J. F. Sauer, B. Threadbare, and Al-Kindi, "The tip5 hash function for recursive starks," Cryptology ePrint Archive, Paper 2023/107, 2023, https://eprint.iacr.org/2023/107. [Online]. Available: https://eprint.iacr.org/2023/107

[35] L. Grassi, D. Khovratovich, R. Lüftenegger, C. Rechberger, M. Schofnegger, and R. Walch, "Hash functions monolith for zk applications: May the speed of sha-3 be with you," Cryptology ePrint Archive, Paper 2023/1025, 2023, https://eprint.iacr.org/2023/1025. [Online]. Available: https://eprint.iacr.org/2023/1025

[36] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," Cryptology ePrint Archive, Paper 2020/315, 2020, https://eprint.iacr.org/2020/315. [Online]. Available: https://eprint.iacr.org/2020/315

[37] E. Ben-Sasson, A. Chiesa, D. Genkin, S. Kfir, E. Tromer, M. Virza, H. Wu, M. Backes, M. Barbosa, A. Chernyakhovsky, D. Fiore, J. Groth, J. A. Kroll, S. MITSUNARI, A. Popovs, R. Reischuk, and T. TERUYA, "libsnark: a c++ library for zksnark proofs," https://github.com/scipr-lab/

libsnark, 2012, SCIPR Lab.

[38] S. Bowe and J. Grigg, "bellman: zk-snark library," https://github.com/zkcrypto/bellman, 2015, zero-knowledge Cryptography in Rust.

[39] A. contributors, "arkworks zksnark ecosystem," 2022, https://arkworks.rs. [Online]. Available: https://arkworks.rs

[40] J. L. Muñoz-Tapia, M. Belles, M. Isabel, A. Rubio, and J. Baylina, "CIRCOM: A Robust and Scalable Language for Building Complex Zero-Knowledge Circuits," *TechRxiv*, 3 2022. [Online]. Available: https://techrxiv.figshare.com/articles/preprint/CIRCOM_A_Robust_and_Scalable_Language_for_Building_Complex_Zero-Knowledge_Circuits/19374986

[41] Iden3, "Circom circuit compiler," GitHub Repository, 'circom', 2022, https://github.com/iden3/circom. [Online]. Available: https://github.com/iden3/circom

[42] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in *ECRYPT hash workshop*, vol. 2007, 2007.

[43] U. Maurer, R. Renner, and C. Holenstein, "Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology," Cryptology ePrint Archive, Paper 2003/161, 2003, https://eprint.iacr.org/2003/161. [Online]. Available: https://eprint.iacr.org/2003/161

[44] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the indifferentiability of the sponge construction," in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 181–197.

[45] R. S. Winternitz, "Producing a one-way hash function from des," in *Advances in Cryptology: Proceedings of Crypto 83*, D. Chaum, Ed. Boston, MA: Springer US, 1984, pp. 203–207. [Online]. Available: https://doi.org/10.1007/978-1-4684-4730-9{_}17

[46] ——, "A secure one-way hash function built from des," in *1984 IEEE Symposium on Security and Privacy*, 1984, pp. 88–88.

[47] J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from pgv," Cryptology ePrint Archive, Paper 2002/066, 2002, https://eprint.iacr.org/2002/066. [Online]. Available: https://eprint.iacr.org/2002/066

[48] D. Wang, "Loopring," https://loopring.org/, 2020, loopring Project Ltd.

[49] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," ZCash Improvement Proposals Website, Sep 2022, https://zips.z.cash. [Online]. Available: https://zips.z.cash

[50] A. Bakhta, E. B. Sasson, A. Levy, and D. L. Gurevich, "Eip-5988: Add poseidon hash function precompile," https://eips.ethereum.org/EIPS/eip-5988, 2022, https://eips.ethereum.org/EIPS/eip-5988.

[51] L. Grassi, R. Lüftenegger, C. Rechberger, D. Rotaru, and M. Schofnegger, "On a generalization of substitution-permutation networks: The hades design strategy," Cryptology ePrint Archive, Paper 2019/1107, 2019, https://eprint.iacr.org/2019/1107. [Online]. Available: https://eprint.iacr.org/2019/1107

[52] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[53] S. Gueron, "Intel advanced encryption standard (aes) new instructions set," 2012.

[54] T. Jakobsen and L. R. Knudsen, "The interpolation attack on block ciphers," in *Fast Software Encryption*, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 28–40.

[55] J.-C. Faugère, P. Gaudry, L. Huot, and G. Renault, "Sub-cubic change of ordering for gröbner basis: A probabilistic approach," in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 170–177. [Online]. Available: https://doi.org/10.1145/2608628.2608669

[56] J. Hoeven and R. Larrieu, "Fast gröbner basis computation and polynomial reduction for generic bivariate ideals," *Applicable Algebra in Engineering, Communication and Computing*, vol. 30, no. 6, pp. 509–539, Dec 2019. [Online]. Available: https://doi.org/10.1007/s00200-019-00389-9

[57] X. Bonnetain, "Collisions on feistel-mimc and univariate gmimc," Cryptology ePrint Archive, Paper 2019/951, 2019, https://eprint.iacr.org/2019/951. [Online]. Available: https://eprint.iacr.org/2019/951

[58] A. Roy, E. Andreeva, and J. F. Sauer, "Interpolation cryptanalysis of unbalanced feistel networks with low degree round functions," Cryptology ePrint Archive, Paper 2021/367, 2021, https://eprint.iacr.org/2021/367. [Online]. Available: https://eprint.iacr.org/2021/367

[59] A. Bariant, C. Bouvier, G. Leurent, and L. Perrin, "Practical Algebraic Attacks against some Arithmetization-oriented Hash Functions," Inria, Research Report, Jan. 2022. [Online]. Available: https://hal.science/hal-03518757

[60] T. Ashur, T. Buschman, and M. Mahzoun, "Algebraic cryptanalysis of hades design strategy: Application to poseidon and poseidon2," Cryptology ePrint Archive, Paper 2023/537, 2023, https://eprint.iacr.org/2023/537. [Online]. Available: https://eprint.iacr.org/2023/537

[61] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash functions based on block ciphers: A synthetic approach," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, ser. Lecture Notes in Computer Science, vol. 773. Springer, 1993, pp. 368–378.

[62] H. Dobbertin, A. Bosselaers, and B. Preneel, "Ripemd-160: A strengthened version of ripemd," in *Fast Software Encryption*, D. Gollmann, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 71–82.

[63] R. C. Merkle, "One way hash functions and des," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed. New York, NY: Springer New York, 1990, pp. 428–446.

[64] S. M. Matyas, "Generating strong one-way functions with cryptographic algorithm," *IBM Technical Disclosure Bulletin*, vol. 27, pp. 5658–5659, 1985.

[65] S. Miyaguchi, K. Ohta, and M. Iwata, "128-bit hash function (n-hash)," *NTT review*, 1990.

[66] B. Preneel, "Analysis and design of cryptographic hash functions," Ph.D. dissertation, Katholieke Universiteit te Leuven Leuven, 1993.

[67] M. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015-08-04 2015.

[68] R. C. Merkle, "Method of providing digital signatures," jan 1982. [Online]. Available: https://patents.google.com/patent/US4309569A/en

[69] ——, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[70] J. C. Hamano, "Git–a stupid content tracker," *Proceedings of the Ottawa Linux Symposium 2006*, vol. 1, pp. 385–394, 2006.

[71] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6. Berkeley, CA, USA, 2003, pp. 68–72.

[72] A. Bakker, "Merkle hash torrent extension," 2009, http://bittorrent.org/beps/bep_0030.html. [Online]. Available: http://bittorrent.org/beps/bep_0030.html

[73] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, apr 2010. [Online]. Available: https://doi.org/10.1145/1773912.1773922

[74] S. Sivasubramanian, "Amazon dynamodb: A seamlessly scalable non-relational database service," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 729–730. [Online]. Available: https://doi.org/10.1145/2213836.2213945

[75] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[76] D. Vujičić, D. Jagodić, and S. Randić, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2018, pp. 1–6.

[77] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *Fast Software Encryption*, B. Roy and W. Meier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 371–388.

[78] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win, "The cipher shark," in *Fast Software Encryption*, D. Gollmann, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 99–111.

[79] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. Elsevier, 1977, vol. 16, pp. 294–306.

[80] L. Grassi, C. Rechberger, and M. Schofnegger, "Proving resistance against infinitely long subspace trails: How to choose the linear layer," *IACR Transactions on Symmetric Cryptology*, vol. 2021, no. 2, pp. 314–352, Jun. 2021. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8913

[81] P. S. L. M. Barreto, B. Lynn, and M. Scott, "Constructing elliptic curves with prescribed embedding degrees," Cryptology ePrint Archive, Paper 2002/088, 2002, https://eprint.iacr.org/2002/088. [Online]. Available: https://eprint.iacr.org/2002/088

[82] B. Edgington, "Bls12-381 for the rest of us," https://hackmd.io/@benjaminion/bls12-381#Resources-and-further-reading, Jun 2023.

[Online]. Available: https://hackmd.io/@benjaminion/bls12-381#Resources-and-further-reading

[83] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," Cryptology ePrint Archive, Paper 2005/133, 2005, https://eprint.iacr.org/2005/133. [Online]. Available: https://eprint.iacr.org/2005/133

[84] J. Wang, "Bn254 for the rest of us," https://hackmd.io/@jpw/bn254, Aug 2022. [Online]. Available: https://hackmd.io/@jpw/bn254

[85] H. M. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, pp. 393–422, 2007, https://www.ams.org/journals/bull/2007-44-03/S0273-0979-07-01153-6/. [Online]. Available: https://www.ams.org/journals/bull/2007-44-03/S0273-0979-07-01153-6/

[86] M. Schofnegger and R. Walch, "Hash functions for zero-knowledge applications zoo," https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo, August 2021, IAIK, Graz University of Technology.

Appendix
comp-col resistance of AO PGV-ELC mode

Proof. We build on the arguments made in the proof of Theorem 1, with the following adjustments:

1) The two colliding inputs $(\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{x}', \boldsymbol{y}')$ are now over $\mathbb{F}_p^{\kappa'} \times \mathbb{F}_p^{n'}$ rather than $\mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n}$.
2) The queries in $\mathcal{Q}$ are now of the kind $Q_i = (\boldsymbol{k}_i, \boldsymbol{m}_i, \boldsymbol{c}_i, b_i)$, where $\boldsymbol{k}_i \in \mathbb{F}_p^{\kappa}$ and $\boldsymbol{m}_i \in \mathbb{F}_p^{n}$.
3) If $\mathcal{A}$ finds a collision, there are $Q_i, Q_j \in \mathcal{Q}$ such that $\boldsymbol{h}_i = \boldsymbol{h}_j$ and:

$$\begin{cases} \boldsymbol{k}_i = \boldsymbol{K}\boldsymbol{x}_i \\ \boldsymbol{m}_i = \boldsymbol{P}\boldsymbol{y}_i \\ \boldsymbol{z}_i = \boldsymbol{F}\boldsymbol{y}_i \\ \boldsymbol{t}_i = \boldsymbol{R}\boldsymbol{c}_i \\ \boldsymbol{h}_i = \boldsymbol{t}_i + \boldsymbol{z}_i \end{cases} \qquad \begin{cases} \boldsymbol{k}_j = \boldsymbol{K}\boldsymbol{x}_j \\ \boldsymbol{m}_j = \boldsymbol{P}\boldsymbol{y}_j \\ \boldsymbol{z}_j = \boldsymbol{F}\boldsymbol{y}_j \\ \boldsymbol{t}_j = \boldsymbol{R}\boldsymbol{c}_j \\ \boldsymbol{h}_j = \boldsymbol{t}_j + \boldsymbol{z}_j \end{cases}$$

4) We extend the notion of *linking*: given $\boldsymbol{v} \in \mathbb{F}_p^{l}$, $\boldsymbol{w} \in \mathbb{F}_p^{n}$ and $\boldsymbol{u} \in \mathbb{F}_p^{n'}$, we now have two kinds of equivalence classes over $\mathbb{F}_p^{l}$, the ones of the kind $[v]_{\boldsymbol{R}}$ with cardinality p^{n-l}, and the ones of the kind $[v]_{\boldsymbol{F}}$ with cardinality $p^{n'-l}$. We now say that $\boldsymbol{u}$ is $\boldsymbol{w}$-$\boldsymbol{v}$-linking (via $\boldsymbol{F}$ and $\boldsymbol{R}$) if $\boldsymbol{R}w + \boldsymbol{F}u = v$.

When either of the first two equations in Item 3 are satisfied, we say respectively that $\boldsymbol{k}_i$ and $\boldsymbol{m}_i$ are *meaningful*. Additionally, $\boldsymbol{c}_i$ is meaningful if both $\boldsymbol{k}_i$ and $\boldsymbol{m}_i$ are meaningful, and if all three of them are meaningful then the query Q_i is meaningful, and we call this event Mean_i. Since $\boldsymbol{K}$ is a left invertible matrix, it is a bijection between $\mathbb{F}_p^{\kappa'}$ and $\mathbb{F}_p^{\kappa}$, hence there are exactly $p^{\kappa'}$ meaningful keys. Analogously, there are $p^{n'}$ meaningful plaintexts $\boldsymbol{m}_i$ for every choice of $\boldsymbol{k}_i$. Note that $\mathcal{A}$ is free to make 'meaningless' queries and exploit them however it likes; nevertheless, at least the two colliding queries must be meaningful. We can conclude that:

$$\Pr[\mathsf{Col}] = \Pr[\exists i, j \leq q\colon (i < j) \wedge \mathsf{Mean}_i \wedge \mathsf{Mean}_j \wedge \mathsf{Link}_{i,j}]$$

where $\mathsf{Link}_{i,j}$ is again the event that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}_j$-linking via $\boldsymbol{R}$ and $\boldsymbol{F}$. We have four cases to consider:

- $b_i = b_j = 0$: the adversary chooses $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ and $\boldsymbol{y}_i$, $\boldsymbol{y}_j$ among at least $p^{\kappa'} - q$ and $p^{n'} - q$ possible values respectively, ensuring that Q_i and Q_j are meaningful. This choice univocally entails the values of $\boldsymbol{k}_i$, $\boldsymbol{k}_j$, $\boldsymbol{m}_i$, $\boldsymbol{m}_j$, $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$. From the right-invertibility of $\boldsymbol{F}$, there are exactly p^{n-l} values of either $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ which map to any specific value of $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$. Since $i < j$, the value of $\boldsymbol{h}_i$ is known to $\mathcal{A}$ when collecting the query Q_j. However, $\boldsymbol{c}_j$ is a random value from a set of cardinality at least $p^n - q$: note that although there are at least 'only' $p^{n'} - q$ meaningful values left, there is no way to know which these are without having already queried them, so the sample space is effectively the whole $\mathbb{F}_p^n$. Since $\boldsymbol{R}$ is right-invertible, the probability that $\boldsymbol{c}_j \in [\boldsymbol{t}_j]$ is at most $\frac{p^{n-l}}{p^n - q} \leq \frac{1}{p^l - q}$, since $l \leq n$. This probability is then precisely the probability of $\boldsymbol{y}_j$ being $\boldsymbol{c}_j$-$\boldsymbol{h}_i$-linking, hence:

$$\Pr[\mathsf{Col}] \leq 1 \cdot 1 \cdot \sum_{j=1}^{q} \sum_{i=1}^{j} \frac{p^{n-l}}{p^n - q} \leq \frac{q^2 + q}{p^l - q}$$

- $b_i = b_j = 1$: the adversary chooses $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, which entails the values of $\boldsymbol{k}_i$ and $\boldsymbol{k}_j$, and also chooses $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$, which are meaningful each with probability at most $p^{n'}/(p^n - q)$. If this is the case, then both $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are random values from sets of size at least $p^{n'} - q$. Since $i < j$, we can assume $\boldsymbol{h}_i$ to be known by $\mathcal{A}$: the probability that $\boldsymbol{y}_j \in [\boldsymbol{z}_j]$ is at most $\frac{p^{n'-l}}{p^{n'} - q} \leq \frac{1}{p^l - q}$ since $l \leq n'$, and this is again the probability of it being $\boldsymbol{c}_j$-$\boldsymbol{h}_i$-linking. Therefore:

$$\Pr[\mathsf{Col}] \leq \frac{p^{n'}}{p^n - q} \cdot \frac{p^{n'}}{p^n - q} \cdot \sum_{j=1}^{q} \sum_{i=1}^{j} \frac{p^{n'-l}}{p^{n'} - q} \leq \frac{q^2 + q}{p^l - q}$$

- $(b_i = 0) \wedge (b_j = 1)$: Same as the previous case, but this time Q_i is always meaningful.
- $(b_i = 1) \wedge (b_j = 0)$: Same as the first case, but this time Q_i is meaningful at most with probability $p^{n'}/(p^n - q)$.

Since the probability of $\mathcal{A}$ finding a collision is independent of its behaviour, the claim is hence proven. □

Appendix
comp-pre resistance of AO PGV-ELC mode

Proof. The probability of finding a preimage is given by:

$$\Pr[\mathsf{Pre}] = \Pr[\exists i \leq q\colon \mathsf{Mean}_i \wedge \mathsf{Link}_i]$$

where Link_i is the event that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}_j$-linking.

- $b_i = 0$: the adversary chooses $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ so that Q_i is meaningful. $\boldsymbol{c}_i$ is then a random element from a set of size at least $p^n - q$, and the probability that $\boldsymbol{y}_i$ is $\boldsymbol{c}_i$-$\boldsymbol{h}$-linking is at most $\frac{p^{n-l}}{p^n - q} \leq \frac{1}{p^l - q}$, hence: $\Pr[\mathsf{Pre}] \leq 1 \cdot \sum_{i=1}^{q} \frac{p^{n-l}}{p^n - q} \leq \frac{q}{p^l - q}$.
- $b_i = 1$: $\boldsymbol{x}_i$ and $\boldsymbol{c}_i$ are chosen arbitrarily, and there is a $p^{n'}/(p^n - q)$ probability that $\boldsymbol{Q}_i$ is meaningful. Even if this is the case, $\boldsymbol{y}_i$ is a random value from a set of size $p^{n'} - q$, and the probability of it being $\boldsymbol{c}_i$-$\boldsymbol{h}$-linking is at most $\frac{p^{n'-l}}{p^{n'} - q} \leq \frac{1}{p^l - q}$, therefore:

$$\Pr[\mathsf{Pre}] \leq \frac{p^{n'}}{p^n - q} \cdot \sum_{i=1}^{q} \frac{p^{n'-l}}{p^{n'} - q} \leq \frac{q}{p^l - q}$$

□

Appendix
HASH-COL Resistance of AO Merkle Tree

Proof. Suppose that we have an adversary $\mathcal{A}$ with access to $\mathcal{C}$, the oracle implementing a random instance of C. After making q queries to $\mathcal{C}$, interleaved with arbitrary computations, $\mathcal{A}$ outputs two messages $M, M' \in \left(\mathbb{F}_p^n\right)^*$, with $M \neq M'$, such that $H_C(M) = H_C(M')$. Let the collision advantage of $\mathcal{A}$ against H_C be $\mathbf{Adv}_{H_C}^{\text{HASH-COL}}(\mathcal{A})$. For any such $\mathcal{A}$, we can build a new adversary $\mathcal{B}$, which achieves the same advantage against C directly, using the same number of queries as $\mathcal{A}$. $\mathcal{B}$ works as follows: first, it runs $\mathcal{A}$ as a sub-routine, obtaining the two messages M and M'. Then, it builds the Merkle trees $\mathcal{T}$ over M and $\mathcal{T}'$ over M'. We can assume w.l.o.g. that the communication tape of $\mathcal{A}$ already contains a record of all the queries to $\mathcal{C}$ that were necessary to build the two trees. If $\boldsymbol{\nu}_0 \neq \boldsymbol{\nu}'_0$, then $\mathcal{A}$ did not actually find a collision, so $\mathcal{B}$ halts rejecting. Otherwise, $\mathcal{B}$ starts matching tuples of the kind $(\boldsymbol{\nu}_i, \boldsymbol{\nu}_{ti+1}, \ldots, \boldsymbol{\nu}_{ti+t})$ from $\mathcal{T}$ with tuples of the kind $\left(\boldsymbol{\nu}'_i, \boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\right)$ from $\mathcal{T}'$. If, at any point in the matching process, it happens that $\boldsymbol{\nu}_i = \boldsymbol{\nu}'_i$ but, for any $j \leq t$, $\boldsymbol{\nu}_{ti+j} \neq \boldsymbol{\nu}'_{ti+j}$, then $\mathcal{B}$ outputs $\left(\boldsymbol{\nu}_{ti+1}, \ldots, \boldsymbol{\nu}_{ti+t}, \boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\right)$, which is a collision for C, and it halts accepting. If the search ends without finding any match, and $|M| = |M|'$, it must be the case that $M = M'$, which is not a valid collision, so $\mathcal{B}$ halts rejecting. Finally, if all children of $\boldsymbol{\nu}_i$ match the children of $\boldsymbol{\nu}'_i$, assuming w.l.o.g. that $|M| < |M|'$, then, for each leaf node $\boldsymbol{\nu}_i \in \mathcal{T}$, it must be the case that $\boldsymbol{\nu}'_i = \boldsymbol{\nu}_i = \boldsymbol{m}_i$. But since $\boldsymbol{\nu}'_i = C\left(\boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\right)$, this means that $\left(\boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\right)$ is actually a preimage for $\boldsymbol{m}_i$. Let Col be the event of $\mathcal{B}$ finding a collision for C and Pre be the event of it finding a preimage instead. From our previous analysis, we have that:

$$\mathbf{Adv}_{H_C}^{\text{HASH-COL}}(\mathcal{A}, q) = \Pr[\mathsf{Col} \vee \mathsf{Pre}] \leq$$
$$\mathbf{Adv}_{C}^{\text{COMP-COL}}(\mathcal{B}, q) + \mathbf{Adv}_{C}^{\text{COMP-PRE}}(\mathcal{B}, q)$$

Since this result does not depend on the behaviour of $\mathcal{A}$, the claim follows. □

Appendix
Backends Performance Comparison

TABLE VI: Time to compute common field operations for target libraries over target fields.

Library	Field	x+y	x+=y	x+=x	x⋆y	x⋆=y	x⋆=x
NTL	BLS-381	50.9 ns	33.2 ns	11.2 ns	152 ns	62.8 ns	45.3 ns
	BN-254	50.8 ns	33.1 ns	11.5 ns	153 ns	59.9 ns	45.2 ns
	Ed-180	78.07 ns	28.5 ns	11.3 ns	75.7 ns	38.6 ns	33.2 ns
libff	BLS-381	8.12 ns	7.12 ns	7.28 ns	22.4 ns	21.6 ns	21.8 ns
	BN-254	8.35 ns	7.46 ns	7.38 ns	19.4 ns	17.7 ns	17.8 ns
	Ed-180	7.93 ns	7.05 ns	7.22 ns	14.0 ns	13.9 ns	13.6 ns
libarith	BLS-381	3.48 ns	2.78 ns	1.80 ns	18.3 ns	18.5 ns	17.8 ns
	BN-254	3.73 ns	2.96 ns	1.76 ns	19.9 ns	20.3 ns	19.7 ns
	Ed-180	2.66 ns	2.19 ns	3.87 ns	12.1 ns	11.5 ns	10.9 ns

In Table VI, we report the time necessary to perform common arithmetic operations over the target backends. In particular, we noticed that `NTL` greatly benefits from in-place operations, due to its use of dynamic allocation, while for `libff` this is less of a concern, as it uses automatic (stack) storage, and for `libarith` the cost is relevant only for addition.

Appendix
ECs and Order of Prime Fields for Experiments

Here we give the different elliptic curves and prime fields used for our experiments.

TABLE VII: Order of the prime fields used in the experiments.

Curve	p	$\log_2(p)$
BLS12-381	`73eda753`...`00000001`	254.86
BN-254	`30644e72`...`f0000001`	253.60
Ed-180	`10357f27`...`80000001`	180.02

Appendix
Parallelized Implementation and Efficiency Comparison

Here we provide a visualization of the results of our parallelized implementations.

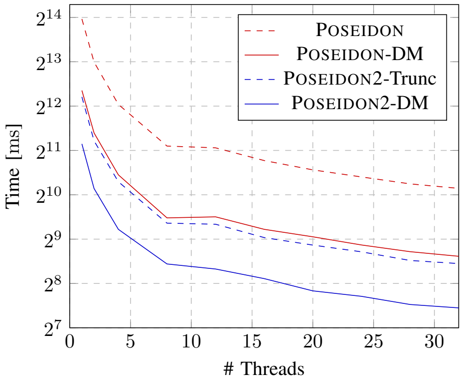

Fig. 5: Multithread scaling of target primitives when building an arity-4 tree (`libff`, BLS-381, $|M| = 2^{20}$).

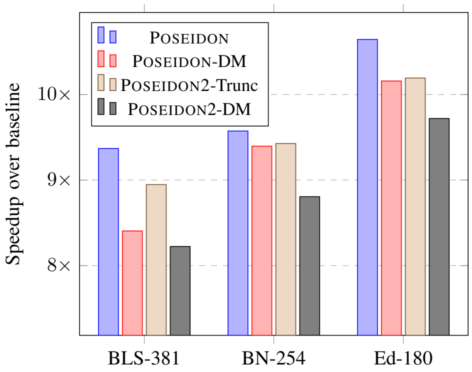

Fig. 6: 16-threads speed-up over serial construction of an arity-4 Merkle tree (`libff`, BLS-381, $|M| = 2^{20}$).