On Efficient and Secure Compression Modes for Arithmetization-Oriented Hashing

Elena Andreeva[1], Rishiraj Bhattacharyya[3], Arnab Roy[2], and Stefano Trevisani[1]

[1] TU Wien, Austria
[2] University of Innsbruck, Austria
[3] University of Birmingham, UK

Abstract. ZK-SNARKs are advanced cryptographic protocols used in private verifiable computation: modern SNARKs allow to encode the invariants of an arithmetic circuit over some large prime field in an appropriate NP language, from which a zero-knowlege short non-interactive argument of knowledge is built. Due to the high cost of proof generation, ZK-SNARKs for large constraint systems are inpractical.
ZK-SNARKs are used in privacy-oriented blockchains such as Filecoin, ZCash and Monero, to verify Merkle tree opening proofs, which in turn requires computing a fixed-input-length (FIL) cryptographic compression function. As classical, bit-oriented hash functions like SHA-2 require huge constraint systems, Arithmetization-Oriented (AO) compression functions have emerged to fill the gap.
Usually, AO compression functions are obtained by applying the Sponge hashing mode on a fixed-key permutation: while this avoids the cost of dynamic key scheduling, AO schedulers are often cheap to compute, making the exploration of AO compression functions based directly on blockciphers a topic of practical interest.
In this work, we first adapt notions related to classical hash functions and their security notions to the AO syntax, and inspired by the classical PGV modes, we propose AO PGV-LC and AO PGV-ELC, two blockcipher-based FIL compression modes with parametrizable input and output sizes. In the ideal cipher model, we prove the collision and preimage resistance of both our modes, and give bounds for collision and opening resistance over Merkle trees of arbitrary arity.
We then experimentally compare the AO PGV-LC mode over the Hades-MiMC blockcipher with its popular Sponge instantiation, Poseidon. The resulting construction, called Poseidon-DM, is 2–5× faster than Poseidon in native computations, and 15–35% faster in generating Merkle tree proofs over the Groth16 SNARK framework, depending on the tree arity. In particular, proof generation for an 8-ary tree over Poseidon-DM is 2.5× faster than for a binary tree with the same capacity over Poseidon. Finally, in an effort to further exploit the benefits of wide trees, we propose a new strategy to obtain a compact R1CS constraint system for Merkle trees with arbitrary arity.

Keywords: Hash function · Block cipher · Arithmetization-Oriented · Merkle tree · Zero-Knowledge · SNARK · Poseidon

1 Introduction

Zero-Knowledge Verifiable Computation. Zero-Knowledge Proof (ZKP) systems [30,29] are advanced cryptographic protocols which allow a prover to convince a verifier about the solvability of some problem without actually revealing a solution. Nowadays, general-purpose ZK *Succinct Non-interactive ARgument of Knowledge* (SNARK) systems [36,50,13,37,28,61] allow the prover to build short proofs which can be quickly checked by any number of verifiers. In particular, given some *bounded computation* that can be represented as an *arithmetic circuit* (i.e. only addition and multiplication gates) over some large prime field $\mathbb{F}_p$, we can encode the invariants of the circuit via some ad-hoc *constraint language*, a process known as *arithmetization*. *Rank-1 Constraint Systems* (R1CS) [22], i.e. systems of bilinear equations, are among the most popular constraint languages, being used in frameworks such as [50,37,12,61]. The main computational bottleneck of modern ZK-SNARK systems is proof generation, whose complexity depends on the size of the underlying constraint system.

One of the most widespread applications of ZK-SNARKs lies in the verification of *Merkle tree opening proofs* [46] (or *authentication paths*), in privacy-preserving blockchains such as ZCash [10], Monero [59,19], and Filecoin [53], among others. In this context, the root of the tree is a *cryptographic commitment* to the contents of the leaves, and it is computed by merging the leaves in a tree-like fashion using a fixed-input-length (FIL) *cryptographic compression function*. To show knowledge of some leaf, the prover must disclose the leaf itself and its co-path, and in order to make this protocol zero-knowledge, it is necessary to arithmetize the underlying compression function. *Classical, bit-oriented* hash functions like SHA-2 [24] require tens of thousands of R1CS constraint in order to be arithmetized (see e.g. [41]), making them impractical for ZK-SNARK applications.

Arithmetization-Oriented Cryptography. Driven by the advancements in ZKP systems, Fully Homomorphic Encryption (FHE) [4] and secure Multi-Party Computation (MPC) [67], in the last decade we have witnessed significant research efforts towards the development of *Arithmetization-Oriented* (AO) cryptography, whose main differences compared to bit-oriented cryptography are the following:

- AO algorithms atomically manipulate elements of large *prime fields* $\mathbb{F}_p$, rather than individual bits, resulting in natural and compact constraint systems over modern SNARK frameworks.
- AO designs are usually parametrizable over the input size, the security parameter, the underlying prime field, and so on, offering a wide choice of concrete instantiations.
- Native execution of AO algorithms is significantly slower than bit-oriented algorithms. A line of recent designs, including `Reinforced Concrete` [32], Tip5 [63] and `Monolith` [33] almost close this gap, but they require specific prime fields and ZK frameworks supporting lookup table arguments [27,20].

– Algebraic cryptanalysis, through techniques like interpolation cryptanalysis [42,56] and Gröbner bases computation [26,40], is the main threat to AO cryptographic primitives.

AO Modes and Provable Security. Many AO primitives, like MiMC [1], GMiMC [2], Poseidon [34], *Rescue Prime* [3], and Arion [57,58], rely on the permutation-based, hashing mode Sponge [14] to achieve both variable-input-length (VIL) and fixed-input-length (FIL) compression. Other designs, such as Griffin [31] and `Anemoi` [17], offer ad-hoc compression modes, which are again based on unkeyed permutations.

The standard way to obtain a secure unkeyed permutation is by fixing the key parameter of a blockcipher to some arbitrary constant (usually 0); this choice has two main consequences for compression: on the one hand, it removes the possibility of fitting part of the message that we want to compress in the key parameter, but on the other hand it allows one to precompute the round keys (since the main key is fixed), avoiding the extra run-time cost induced by the key scheduling algorithm. In classical symmetric cryptography this is generally a winning trade-off, as key scheduling can be as much (if not more) expensive than encryption itself, like in the case of AES [54,38], hence computing, say, two or more calls to a fixed-key blockcipher with block size n, or one call to a fixed-key blockcipher with block size $m > n$ can be faster than computing one call to a (non-fixed-key) blockcipher with block size n.

In AO cryptography, with some notable exceptions such as the MARVELlous family of blockciphers [3,5], the key scheduling algorithm is extremely lightweight, typically being a linear or an affine transformation of the master key. Some constructions, like MiMC, even allowed the use of no scheduler at all, although this choice was then shown to be vulnerable to slide attacks [16]. In fact, when compared to the cost of the non-linear operations involved in AO encryption, affine schedulers can be computed almost for free.

Many well-known modes for building secure compression and hash functions from blockciphers are described by the Preneel-Govaerts-Vandewalle (PGV) framework [52]. Although the PGV modes are inherently related to the Merkle-Damgård (MD) hashing paradigm [47,23], and are defined in the bit-oriented setting, we extracted and generalized the underlying compression modes in order to be used in the Merkle tree setting.

More specifically, our proposed mode, called AO PGV-LC, can be seen as an adaptation of the compression function which underlies the Davies-Meyer [66] and Matyas-Meyer-Oseas [44] iterated modes to the AO setting, as well as an extension which allows more flexibility in the output size. We also propose a further extension, called AO PGV-ELC, which also allows for more flexible input sizes, making it a good choice when the blockcipher's plaintext and key sizes do not perfectly match the required input size.

Suppose that we want to compress some message consisting of two field elements m_1 and m_2 over $\mathbb{F}_p$. If $p \approx 2^{256}$, to get ≈ 128 bits of security we could use a fixed-key blockcipher with a block size of three field elements in Sponge mode, where the extra element is kept for the capacity [14], and output the first

element of the permuted message. On the other hand, we could achieve the same result, without compromising in security, by using a (non-fixed-key) blockcipher with a block size of just 1 field element in AO PGV-LC mode. As we said, AO designs are in general very flexible with respect to their block size n and key size $\kappa = n$: if we assume that the computational complexity of the blockcipher design scales linearly with n, with some slope s, we could expect a theoretical efficiency improvement of $\approx s\frac{2n+1}{n}$ when using the AO PGV-LC mode compared to using a Sponge mode.

The switch from bit-oriented to AO treatment, together with the proposed extensions, mandates the investigation of the relevant security properties from scratch. In particular, we focus on the properties of *collision resistance* (i.e. the problem of finding two inputs which produce the same output) and *preimage resistance* (i.e. the problem of finding some input which produces a given output). These properties, which are fundamental requirements for any cryptographic compression mode, are also required for modular proofs of higher level structures.

In the context of Merkle tree membership proofs, it is important that the Merkle tree itself is *collision resistant*, so that it is hard to find inputs producing the same commitment. In addition, the Merkle tree should also be *opening proof resistant*, so that it is hard to forge a false proof of membership. While these two latter results are well-known for binary trees in the bit-oriented setting, we define and prove them in the AO setting and for trees of arbitrary arity.

Our Contributions

In this work we make the following contributions:

1. AO Syntax and Security Definitions. In Section 2 we adapt the classical syntax for blockcipher and hash function primitives to the AO setting. We give the relevant AO definitions for well-established modular constructive approaches. These include the PGV-MD iterated compression functions, the VIL AO Sponge mode of hashing, which are provided to highlight the differences with our new mode, and the VIL AO Merkle tree mode of hashing. We further tailor the formal security definitions of collision and preimage resistance of hash functions and compression functions to the general AO context. To support the application of Merkle trees in ZK-SNARKs, we provide a dedicated proof opening resistance definition.
2. Two new modes of compression. In Section 3, we propose the new AO PGV-LC mode of FIL compression over blockciphers: given two inputs of size equal to the key size κ and the block size n, they are processed by the underlying blockcipher, mixed with an appropriate feedback value, and then compressed to an arbitrary output size $l < n$ by an arbitrary matrix that must satisfy some basic properties. From AO PGV-LC, we derive a further generalization which we call AO PGV-ELC, that also allows for the two inputs to have size $\kappa' < \kappa$ and $n' < n$ respectively. We opted for this two-step generalization process as AO PGV-ELC requires a more careful analysis to show security, and a secure parametrization is harder to obtain; on the other hand, AO PGV-LC covers a wide range of use-cases.

3. Security proofs for the new modes. In Section 4, we provide security results for the two new modes. Namely, we show that both AO PGV-LC and AO PGV-ELC are collision resistant up to $\left(q^2+q\right)/\left(p^l-q\right)$ queries to the underlying blockcipher, where l is the number of output elements over $\mathbb{F}_p$, and preimage resistant up to $q/\left(p^l-q\right)$ queries.
   For t-ary AO Merkle trees, we reduce collision and opening proof forgery resistance to the collision and preimage resistance of the underlying compression function. Our proofs are generic and enable secure instantiations with sound AO blockciphers, namely, they allow the bulk of cryptanalysis to be shifted to the underlying cipher rather than the hash function.
4. Experiments and t-ary Merkle Tree R1CS. In Section 5, we consider the widely adopted Sponge hash function Poseidon [34,53,65,6], which was built on top of the Hades-MiMC [35] blockcipher with a fixed key. We weight it up against the same blockcipher (this time without fixing the key) instantiated the AO PGV-LC mode, which we call Poseidon-DM, as the resulting compression function is similar in structure to the Davies-Meyer iteration function. Our results show that Poseidon-DM achieves up to 5× higher throughput when used in native computations to build a Merkle tree, both in serial and parallel code, and up to 35% faster proof generation time for a Merkle tree opening in the Groth16 framework, a popular R1CS-based ZK-SNARK system (see [11,18]). We demonstrate that for Poseidon there are important performance benefits derived from using Merkle trees with arity $t > 2$ (for example, we measured a 2.5× speed-up when every node has 4 children rather than 2). In this direction, since the best arity depends on the concrete compression function that one is using, and due to the apparent lack of publicly available implementations, we also propose a novel R1CS for authentication paths over trees of arbitrary arity, which we optimize to be as compact as possible. Indeed, by combining the optimal choice of arity for the Merkle tree with our proposed compression modes, we get ≈ 6× faster native Merkle tree build time and ≈ 3× faster SNARK proof generation time compared to the standard Poseidon over binary Merkle trees.

2 Preliminaries

2.1 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* $\mathrm{char}(\mathbb{F}_p) = p$, 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 n-dimensional *vector space* 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 $\mathbb{F}_p$. We consider elements of $\mathbb{F}_p^n$ to be *column vectors*, and the dual $\left(\mathbb{F}_p^n\right)^\intercal$ to be the space of *row vectors*. 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 matrix where all the entries in the main diagonal have value 1 and all other entries have value 0. The transpose of a vector $\boldsymbol{a}$ (resp. a matrix $\boldsymbol{A}$) is denoted with $\boldsymbol{a}^\intercal$ (resp. $\boldsymbol{A}^\intercal$).

Remark 2.1. Most of the definitions given in this section are already 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 2.1 (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 $\left\{E_{\boldsymbol{k}}^{-1}\right\}$ 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 $\left\{E_{\boldsymbol{k}}^{-1}\right\}$.

Definition 2.2 (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 one single argument $\boldsymbol{x} \in \mathbb{F}_p^{ml}$. A compression function is a Fixed-Input-Length function (FIL), usually with a 'small' input size.

Definition 2.3 (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 have well-behaved padding functions, even more so in AO cryptography where there is not 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 an appropriate padding function is available to us when needed.

2.2 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 [52]. The PGV modes are tightly related to the Merkle-Damgård (MD) mode of hashing [47], in that they generalize well-known modes like Davies-Meyer [66], Matyas-Meyer-Oseas [44], or Miyaguchi-Preneel [48,51], 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 2.4 (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 [14]. Rather than using a blockcipher as the underlying primitive, the Sponge mode operates over an unkeyed permutation.

Definition 2.5 (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:*

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

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:

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

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) [25]: 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 [45,46], 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 2.6 (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 $\forall i < n$:*

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

Given a padding function $\mathrm{Pad}\colon (\mathbb{F}_p)^* \to \left\{M \in \left(\mathbb{F}_p^l\right)^* \mid \exists h \in \mathbb{N}\colon |M| = t^h\right\}$*, 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 [39], P2P networks [21,7], database systems [43,62], and blockchains [49,64].

2.3 Security Notions

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 E_{E,q}(k, m, b)
    static i ← 0
    if i ≥ q then
        return ⊥
    i ← i + 1
    if b = 0 then
        return E_k(m)
    return E_k^{-1}(m)
```

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 2.2. 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 2.7 (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 is:*

$$\mathbf{Adv}^{\text{COMP-COL}}_{C_E}(\mathcal{A}, q) = \Pr\Big[(\boldsymbol{y}, \boldsymbol{y}') \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_{E,q}}() \colon \boldsymbol{y} \neq \boldsymbol{y}' \wedge C_E(\boldsymbol{y}) = C_E(\boldsymbol{y}')\Big]$$

Definition 2.8 (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 is:*

$$\mathbf{Adv}^{\text{COMP-PRE}}_{C_E}(\mathcal{A}, q) = \Pr\Big[\boldsymbol{y} \xleftarrow{\$} \mathbb{F}_p^n, \boldsymbol{x} \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_{E,q}}(\boldsymbol{y}) \colon C_E(\boldsymbol{x}) = \boldsymbol{y}\Big]$$

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 [55].

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 2.9 (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 2.10 (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 $(\mathcal{G}, \mathcal{V})$ is:*

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

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

3 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 3.1 (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 4, 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.

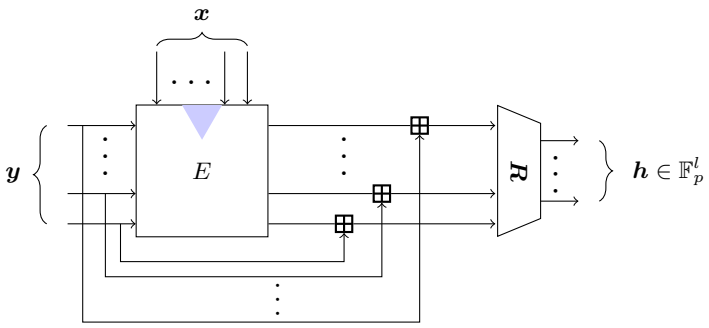

Fig. 1. AO PGV-LC mode over a κ-n-elements blockcipher E, inputs $\boldsymbol{x} \in \mathbb{F}_p^{\kappa}$ and $\boldsymbol{y} \in \mathbb{F}_p^{n}$, right-invertible matrix $\boldsymbol{R} \in \mathbb{F}_p^{l \times n}$, and output $\boldsymbol{h} \in \mathbb{F}_p^{l}$, where $n > l \geq 1$.

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

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

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 4. A pictorial representation of the AO PGV-ELC mode is given in Figure 2.

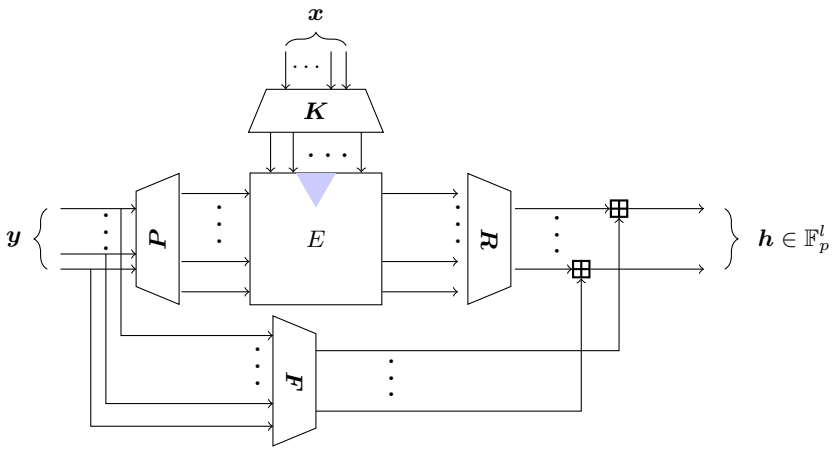

Fig. 2. AO PGV-ELC mode over a κ-n-elements blockcipher E, inputs in $\boldsymbol{x} \in \mathbb{F}_p^{\kappa'}$ and $\boldsymbol{y} \in \mathbb{F}_p^{n'}$, left invertible matrices $\boldsymbol{K} \in \mathbb{F}_p^{\kappa \times \kappa'}$ and $\boldsymbol{P} \in \mathbb{F}_p^{n \times n'}$, right-invertible matrices $\boldsymbol{F} \in \mathbb{F}_p^{l \times n'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{l \times n}$, and output $\boldsymbol{h} \in \mathbb{F}_p^l$, where $1 \leq l \leq n' \leq n$ and $\kappa' \leq \kappa$.

4 Security Proofs

In [15], 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 1, 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.

Table 1. The AO equivalent of the 12 Group-1 PGV compression modes of [15]. 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	$C^{(\iota)}(\boldsymbol{x}, \boldsymbol{y})$
1	$E_{\boldsymbol{x}}(\boldsymbol{y}) + \boldsymbol{y}$
2	$E_{\boldsymbol{x}}(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{x} + \boldsymbol{y}$
3	$E_{\boldsymbol{x}}(\boldsymbol{y}) + \boldsymbol{x} + \boldsymbol{y}$
4	$E_{\boldsymbol{x}}(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{y}$

ι	$C^{(\iota)}(\boldsymbol{x}, \boldsymbol{y})$
5	$E_{\boldsymbol{y}}(\boldsymbol{x}) + \boldsymbol{x}$
6	$E_{\boldsymbol{y}}(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{x} + \boldsymbol{y}$
7	$E_{\boldsymbol{y}}(\boldsymbol{x}) + \boldsymbol{x} + \boldsymbol{y}$
8	$E_{\boldsymbol{y}}(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{x}$

ι	$C^{(\iota)}(\boldsymbol{x}, \boldsymbol{y})$
9	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{y}) + \boldsymbol{y}$
10	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{x}) + \boldsymbol{x}$
11	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{y}) + \boldsymbol{x}$
12	$E_{\boldsymbol{x}+\boldsymbol{y}}(\boldsymbol{x}) + \boldsymbol{y}$

In the design phase of the AO PGV-LC and the AO 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 [15] is enough to guarantee the security of the AO PGV-LC mode.

4.1 Security of AO PGV-LC mode

Theorem 4.1 (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:*

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

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, $\Pr[\mathsf{Col}] = \mathbf{Adv}^{\text{COMP-COL}}_{C_{E,\boldsymbol{R}}}(\mathcal{A})$. 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 $\mathcal{Q} = (Q_i)_{i \in \{1,\ldots,q\}}$, 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:

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

We have four cases to consider, one for 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, $\Pr[\mathsf{Col}] \leq \big(q^2 + q\big)/\big(p^l - q\big)$.
- $b_i = 0 = 1 - b_j$: 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 $\Pr[\mathsf{Col}] \leq \big(q^2 + q\big)/\big(p^l - q\big)$.
- $b_j = 0 = 1 - b_i$: 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 4.2 (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 4.1. 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 $\Pr[\mathsf{Pre}] \leq q/\big(p^l - q\big)$.

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

4.2 Security of AO PGV-ELC mode

The main difference between the AO PGV-LC and the AO PGV-ELC 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 4.3 (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 3.2, and a number of queries $q < p^l$, it holds that:*

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

Proof. We build on the arguments made in the proof of Theorem 4.1, 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. □

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

Theorem 4.4 (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 3.2, it holds that:*

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

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:

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

□

4.3 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 4.5 (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.

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 $\mathbf{Adv}^{\text{HASH-COL}}_{H_C}(\mathcal{A})$. 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 $\boldsymbol{\nu}'_i = C\big(\boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\big)$, this means that $\big(\boldsymbol{\nu}'_{ti+1}, \ldots, \boldsymbol{\nu}'_{ti+t}\big)$ 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:

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

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

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 $M \in \big(\mathbb{F}_p^n\big)^{t^h}$ (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 $\forall j < h$:

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

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

Theorem 4.6 (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:*

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

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 $M \in \big(\mathbb{F}_p^n\big)^{t^h}$, 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 $\tilde{\pi} = (i, \tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{ht'})$ 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 $(d_{h-1}, \ldots, d_0)$ of i and starts matching, for $j \in \{h-1, \ldots, 0\}$, $\boldsymbol{c}_j$ with $\tilde{\boldsymbol{c}}_j$ and

$\boldsymbol{x}_{jt'+1}, \ldots, \boldsymbol{x}_{jt'+t'}$ with $\tilde{\boldsymbol{x}}_{jt'+1}, \ldots, \tilde{\boldsymbol{x}}_{jt'+t'}$: if the match is only partial, then the two vectors:

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

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. □

5 Implementation and Experiments

In order to compare AO PGV-LC with Sponge, among the many available arithmetization-oriented constructions, we decided to select the HADES-MiMC blockcipher design [35]: firstly, the design itself, being based on a variation of substitution-permutation networks (SPN), has undergone an acceptable amount of cryptanalysis; secondly, the Sponge hash function derived from HADES-MiMC, i.e. POSEIDON [34], is already deployed in industry applications (e.g. Filecoin); finally, it is well-defined for any arbitrary block size (although a block size of 1 field element is a special case, see Remark 5.1).

Definition 5.1 (POSEIDON-DM compression function). *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 [35]. Then, given some* $m < n$*, we call* POSEIDON-DM *the compression function:*

$$C(\boldsymbol{x}, \boldsymbol{y})\colon \mathbb{F}_p^n \times \mathbb{F}_p^n \to \mathbb{F}_p^m = \lfloor C_E^{(5)}(\boldsymbol{x}, \boldsymbol{y}) \rfloor_m$$

Remark 5.1. The HADES-MiMC blockcipher design 'collapses' in the MiMC construction [1] when the block size is of just one field element, nullifying the benefits of the partial SPN structure that gives its efficiency. For this reason, we will mark with an asterisk the results concerning 2-to-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 39980 instance. For the native performance comparison part, we used the C++ library `libff`[4] for the finite field arithmetic operations. For the ZK-SNARK comparisons, we implemented the R1CS constraint systems in the C++ library `libsnark`[5], which offers an implementation

[4] `https://github.com/scipr-lab/libff`
[5] `https://github.com/scipr-lab/libsnark`

of the Groth16 [37] ZK-SNARK framework. All code was compiled with the Intel® oneAPI DPC++ Compiler 2023.2 with compiler flags `-std=c++17 -Ofast -march=native` for serial code, and all previous flags plus the `-fopenmp` flag for parallel code. As target prime field, we used the scalar field of the BLS12–381 elliptic curve [8].

5.1 Native Performance

We start by comparing the native performance of Poseidon with Poseidon-DM. Given a prime p, an arity t, a height h, and a message $M \in (\mathbb{F}_p)^*$ such that $|M| = t^h$, we measure the time it takes to build t-ary Merkle tree $\mathcal{T}_{C,M}$, when C is instantiated with t-to-1 Poseidon or with t-to-1 Poseidon-DM.

Table 2. Single-thread build time for a Merkle tree of varying arity and input message length using the target compression functions in the BLS12 curve.

		MT Arity			
	$\|M\|$	2:1*	2:1	4:1	8:1
Poseidon	2^6	\	1.48 ms	1.41 ms	1.72 ms
	2^{12}	\	0.087 s	0.079 s	0.105 s
	2^{18}	\	5.44 s	4.98 s	6.62 s
Poseidon-DM	2^6	0.43 ms	0.78 ms	0.28 ms	0.35 ms
	2^{12}	0.024 s	0.046 s	0.016 s	0.020 s
	2^{18}	1.52 s	2.92 s	0.97 s	1.28 s
Speed-up	2^6	\	1.90×	4.98×	4.86×
	2^{12}	\	1.89×	5.05×	5.13×
	2^{18}	\	1.86×	5.14×	5.16×

In Table 2 we report our results: even for 2:1 compression, albeit unable to take advantage of the partial-SPN structure, Poseidon-DM is already 3.5× faster than Poseidon, confirming that the cost of the key schedule is indeed negligible, and vastly compensated by the reduced state size. Extending our analysis to different arities, we have two interesting results: first, the gap between the two constructions increases even more, with Poseidon-DM being roughly 5× faster than Poseidon; secondly, it turns out that it is extremely important to find the correct arity to maximize efficiency: to Merkle hash a message of length 2^{12} field elements, we need either 4095 calls to a 2:1 compression function, 1365 calls to a 4:1 compression function, or 585 calls to an 8:1 compression function. In the specific case of Poseidon and Poseidon-DM, we have a sweet spot at a compression rate of 4:1 elements. By combining these two results, we obtain a 6× speed-up w.r.t. 2:1 Poseidon, the most popular instance in real-world applications. We also measured how well the performance scales when the tree is being built by multiple threads concurrently, and the results are reported in

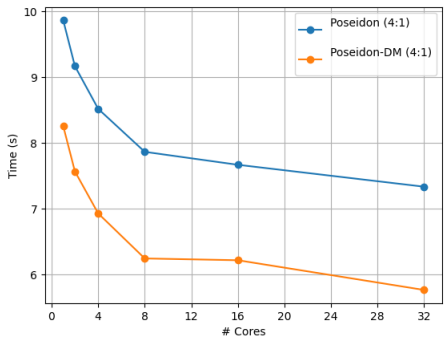

Fig. 3. Multithread performance scaling of Poseidon and Poseidon-DM when building a tree of arity 4 over a message of length 2^{20}.

Figure 3. Both functions scale similarly, with a dip at 16 cores likely caused by the P-cores/E-cores architecture of the specific CPU; interestingly, memory bandwidth does not seem to be much of a problem, and even from 16 to 32 threads (8 of which are due to hyper-threading) we still get a 60% speed-up.

5.2 ZK-SNARK Performance

The main bottleneck of ZK-SNARK frameworks usually lies in the generation of the proof, therefore we target proof generation time as our efficiency metric. In turn, the complexity of building a proof varies on the proving framework itself: in ZK-STARK systems [9], it fundamentally depends on the depth of the arithmetic circuit. In $\mathcal{P}\mathfrak{lon}\mathcal{K}$-like systems, it depends on the kind of constraints which are used, as well as on their number. For ZK-SNARK systems based on QAPs (and in turn on R1CS), it fundamentally depends on the size of the constraint system itself (although it might also depend on its sparsity): a lower number of constraints is normally directly related to an improvement in the measured performance. In Table 3, we can see that Poseidon-DM always requires fewer

Table 3. Number of R1CS constraints for Poseidon and Poseidon-DM.

	MT Arity			
	2:1*	2:1	4:1	8:1
Poseidon	\	258	306	402
Poseidon-DM	330	231	231	282

constraints than Poseidon, and the gap gets wider for larger state sizes, up to 40% in the case of 8:1 compression. Once again, we can see how increasing the arity improves proof generation time all over the board: for example, generating

Table 4. Proof generation time in the Groth16 framework (`libsnark`) for a Merkle tree opening over messages of varying length and different arities.

		MT Arity			
	$\|M\|$	2:1*	2:1	4:1	8:1
Poseidon	2^6	\	0.33 s	0.17 s	0.13 s
	2^{12}	\	0.61 s	0.38 s	0.31 s
	2^{18}	\	0.89 s	0.54 s	0.47 s
	2^{24}	\	1.19 s	0.73 s	0.62 s
	2^{30}	\	1.48 s	0.89 s	0.79 s
Poseidon-DM	2^6	0.39 s	0.29 s	0.14 s	0.093 s
	2^{12}	0.75 s	0.55 s	0.29 s	0.23 s
	2^{18}	1.10 s	0.80 s	0.42 s	0.35 s
	2^{24}	1.44 s	1.05 s	0.56 s	0.46 s
	2^{30}	1.73 s	1.31 s	0.70 s	0.58 s
Average Speed-up		\	+13%	+28%	+35%

a proof for 8:1 Poseidon-DM is approximately 2.5× faster than for 2:1 Poseidon. About concrete timings, we expect them to closely follow the number of constraints, and this is indeed confirmed by Table 4.

5.3 Optimized R1CS for t-ary Merkle Tree

As we mentioned in the introduction, R1CS systems constrain the computation by means of a system of bilinear equations of the kind $(\boldsymbol{A}\boldsymbol{x})\odot(\boldsymbol{B}\boldsymbol{x}) = \boldsymbol{C}\boldsymbol{x}$ (where $\odot$ denotes the Hadamard product). It is well known how to build a R1CS system for *binary* Merkle trees[6]; however, the only public implementation that we found that also offers wider arities is [60], and even there, the constraint systems are hardcoded for $t \in \{2, 4, 8\}$. While writing our own R1CS for an arbitrary t, we found that a small change in the classical opening proof protocol (described in Section 4.3) allows for a more compact R1CS.

In the binary tree case, 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:

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

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

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

One possible way to generalize to any arity $t \geq 2$, similar to the one used in [60], 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 semantically equivalent to the following:

$$\forall 0 \leq j < h\colon \begin{cases} \forall 1 \leq k \leq b\colon d_{j,k} \cdot (1 - d_{j,k}) = 0 \\ \prod_{k=1}^{b} (1 - d_{j,k}) = s_{j,1} \\ \ldots \\ \prod_{k=1}^{b} d_{j,k} = s_{j,t} \\ s_{j,1} \cdot \boldsymbol{c}_j + \tilde{s}_{j,1} \cdot \boldsymbol{x}_{jt'+1} = \boldsymbol{y}_{j,1} \\ s_{j,t} \cdot \boldsymbol{c}_j + \tilde{s}_{j,t} \cdot \boldsymbol{x}_{jt'+t'} = \boldsymbol{y}_{j,t} \\ \forall 2 \leq k < t\colon (s_{j,k} \cdot \boldsymbol{c}_j) + (s_{j,k-1} \cdot \boldsymbol{x}_{jt'+k-1}) + (\tilde{s}_{j,k} \cdot \boldsymbol{x}_{jt'+k}) = \boldsymbol{y}_{j,k} \\ \boldsymbol{c}_{j+1} = C(\boldsymbol{y}_{j,1}, \ldots, \boldsymbol{y}_{j,t}) \end{cases}$$

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.

Table 5. Comparison of the number of R1CS constraints in the unoptimized and optimized Merkle Tree circuits over the Poseidon-DM compression function, for trees containing 2^{24} nodes, at various arities and node size n.

		MT Arity		
	n	4:1	8:1	16:1
Unoptimized	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
Speed-up	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%

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:

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

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. In Table 5, we show the concrete improvement over Merkle Trees of different arities and node sizes where C is instantiated by Poseidon-DM: with compression functions requiring fewer constraints [31,17,58], we expect the gap to be even more noticeable.

Acknowledgements

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. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 191–219. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
2. Albrecht, M.R., Grassi, L., Perrin, L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel structures for mpc, and more. Cryptology ePrint Archive, Paper 2019/397 (2019), https://eprint.iacr.org/2019/397, https://eprint.iacr.org/2019/397
3. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Paper 2019/426 (2019). https://doi.org/10.13154/tosc.v2020.i3.1-45, https://eprint.iacr.org/2019/426, https://eprint.iacr.org/2019/426
4. Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C.A., Strand, M.: A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192 (2015), https://eprint.iacr.org/2015/1192, https://eprint.iacr.org/2015/1192

5. Ashur, T., Dhooghe, S.: Marvellous: a stark-friendly family of cryptographic primitives. Cryptology ePrint Archive, Paper 2018/1098 (2018), https://eprint.iacr.org/2018/1098, https://eprint.iacr.org/2018/1098
6. Bakhta, A., Sasson, E.B., Levy, A., Gurevich, D.L.: Eip-5988: Add poseidon hash function precompile. https://eips.ethereum.org/EIPS/eip-5988 (2022), https://eips.ethereum.org/EIPS/eip-5988
7. Bakker, A.: Merkle hash torrent extension (2009), http://bittorrent.org/beps/bep_0030.html, http://bittorrent.org/beps/bep_0030.html
8. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. Cryptology ePrint Archive, Paper 2002/088 (2002), https://eprint.iacr.org/2002/088, https://eprint.iacr.org/2002/088
9. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018), https://eprint.iacr.org/2018/046, https://eprint.iacr.org/2018/046
10. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
11. Ben-Sasson, E., Chiesa, A., Genkin, D., Kfir, S., Tromer, E., Virza, M., Wu, H., Backes, M., Barbosa, M., Chernyakhovsky, A., Fiore, D., Groth, J., Kroll, J.A., MITSUNARI, S., Popovs, A., Reischuk, R., TERUYA, T.: libsnark: a c++ library for zksnark proofs. https://github.com/scipr-lab/libsnark (2012), SCIPR Lab
12. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. Cryptology ePrint Archive, Paper 2018/828 (2018), https://eprint.iacr.org/2018/828, https://eprint.iacr.org/2018/828
13. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Paper 2013/879 (2013), https://eprint.iacr.org/2013/879, https://eprint.iacr.org/2013/879
14. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. vol. 2007 (2007)
15. Black, J., Rogaway, P., Shrimpton, T.: 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, https://eprint.iacr.org/2002/066
16. Bonnetain, X.: Collisions on feistel-mimc and univariate gmimc. Cryptology ePrint Archive, Paper 2019/951 (2019), https://eprint.iacr.org/2019/951, https://eprint.iacr.org/2019/951
17. Bouvier, C., Briaud, P., Chaidos, P., Perrin, L., Salen, R., Velichkov, V., Willems, D.: New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and jive compression mode. Cryptology ePrint Archive, Paper 2022/840 (2022), https://eprint.iacr.org/2022/840, https://eprint.iacr.org/2022/840
18. Bowe, S., Grigg, J.: bellman: zk-snark library. https://github.com/zkcrypto/bellman (2015), zero-knowledge Cryptography in Rust
19. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066 (2017), https://eprint.iacr.org/2017/1066, https://eprint.iacr.org/2017/1066

20. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Paper 2019/1076 (2019), https://eprint.iacr.org/2019/1076, https://eprint.iacr.org/2019/1076
21. Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer systems. vol. 6, pp. 68–72. Berkeley, CA, USA (2003)
22. Cramer, R., Damgård, I.: Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In: Krawczyk, H. (ed.) Advances in Cryptology — CRYPTO '98. pp. 424–441. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
23. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 416–427. Springer New York, New York, NY (1990)
24. Dang, Q.H.: Secure Hash Standard. National Institute of Standards and Technology (Jul 2015). https://doi.org/10.6028/nist.fips.180-4, http://dx.doi.org/10.6028/NIST.FIPS.180-4
25. Dworkin, M.: Sha-3 standard: Permutation-based hash and extendable-output functions (2015-08-04 2015). https://doi.org/https://doi.org/10.6028/NIST.FIPS.202
26. Faugère, J.C., Gaudry, P., Huot, L., Renault, G.: Sub-cubic change of ordering for gröbner basis: A probabilistic approach. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. pp. 170–177. ISSAC '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2608628.2608669, https://doi.org/10.1145/2608628.2608669
27. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315 (2020), https://eprint.iacr.org/2020/315, https://eprint.iacr.org/2020/315
28. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019), https://eprint.iacr.org/2019/953, https://eprint.iacr.org/2019/953
29. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. J. ACM **38**(3), 690–728 (jul 1991). https://doi.org/10.1145/116825.116852, https://doi.org/10.1145/116825.116852
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012, https://doi.org/10.1137/0218012
31. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst meets fluid-spn: Griffin for zero-knowledge applications. Cryptology ePrint Archive, Paper 2022/403 (2022), https://eprint.iacr.org/2022/403, https://eprint.iacr.org/2022/403
32. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Reinforced concrete: A fast hash function for verifiable computation. Cryptology ePrint Archive, Paper 2021/1038 (2021). https://doi.org/10.1145/3548606.3560686, https://eprint.iacr.org/2021/1038, https://eprint.iacr.org/2021/1038
33. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: 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, https://eprint.iacr.org/2023/1025

34. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458 (2019), https://eprint.iacr.org/2019/458, https://eprint.iacr.org/2019/458
35. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schofnegger, M.: On a generalization of substitution-permutation networks: The hades design strategy. Cryptology ePrint Archive, Paper 2019/1107 (2019), https://eprint.iacr.org/2019/1107, https://eprint.iacr.org/2019/1107
36. Groth, J.: 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. Lecture Notes in Computer Science, vol. 6477, pp. 341–358. Springer (2010). https://doi.org/10.1007/978-3-642-17373-8_20, https://www.iacr.org/archive/asiacrypt2010/6477343/6477343.pdf
37. Groth, J.: On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260 (2016), https://eprint.iacr.org/2016/260, https://eprint.iacr.org/2016/260
38. Gueron, S.: Intel advanced encryption standard (aes) new instructions set (2012)
39. Hamano, J.C.: Git–a stupid content tracker. Proceedings of the Ottawa Linux Symposium 2006 **1**, 385–394 (2006)
40. Hoeven, J., Larrieu, R.: Fast gröbner basis computation and polynomial reduction for generic bivariate ideals. Applicable Algebra in Engineering, Communication and Computing **30**(6), 509–539 (Dec 2019). https://doi.org/10.1007/s00200-019-00389-9, https://doi.org/10.1007/s00200-019-00389-9
41. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification. ZCash Improvement Proposals Website (Sep 2022), https://zips.z.cash, https://zips.z.cash
42. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) Fast Software Encryption. pp. 28–40. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
43. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. **44**(2), 35–40 (apr 2010). https://doi.org/10.1145/1773912.1773922, https://doi.org/10.1145/1773912.1773922
44. Matyas, S.M.: Generating strong one-way functions with cryptographic algorithm. IBM Technical Disclosure Bulletin **27**, 5658–5659 (1985)
45. Merkle, R.C.: Method of providing digital signatures (jan 1982), https://patents.google.com/patent/US4309569A/en
46. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology — CRYPTO '87. pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
47. Merkle, R.C.: One way hash functions and des. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 428–446. Springer New York, New York, NY (1990)
48. Miyaguchi, S., Ohta, K., Iwata, M.: 128-bit hash function (n-hash). NTT review (1990)
49. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
50. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly practical verifiable computation. Cryptology ePrint Archive, Paper 2013/279 (2013), https://eprint.iacr.org/2013/279, https://eprint.iacr.org/2013/279

51. Preneel, B.: Analysis and design of cryptographic hash functions. Ph.D. thesis, Katholieke Universiteit te Leuven Leuven (1993)
52. Preneel, B., Govaerts, R., Vandewalle, J.: 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. Lecture Notes in Computer Science, vol. 773, pp. 368–378. Springer (1993). https://doi.org/10.1007/3-540-48329-2_31
53. Psaras, Y., Dias, D.: 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). pp. 80–80 (2020). https://doi.org/10.1109/DSN-S50200.2020.00043
54. Rijmen, V., Daemen, J., Preneel, B., Bosselaers, A., De Win, E.: The cipher shark. In: Gollmann, D. (ed.) Fast Software Encryption. pp. 99–111. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
55. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: Roy, B., Meier, W. (eds.) Fast Software Encryption. pp. 371–388. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
56. Roy, A., Andreeva, E., Sauer, J.F.: Interpolation cryptanalysis of unbalanced feistel networks with low degree round functions. Cryptology ePrint Archive, Paper 2021/367 (2021), https://eprint.iacr.org/2021/367, https://eprint.iacr.org/2021/367
57. Roy, A., Steiner, M.: Generalized triangular dynamical system: An algebraic system for constructing cryptographic permutations over finite fields (2022). https://doi.org/10.48550/ARXIV.2204.01802, https://arxiv.org/abs/2204.01802, https://arxiv.org/abs/2204.01802
58. Roy, A., Steiner, M.J., Trevisani, S.: Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems (2023)
59. van Saberhagen, N.: Cryptonote v 2.0 (2013), https://api.semanticscholar.org/CorpusID:2711472
60. Schofnegger, M., Walch, R.: Hash functions for zero-knowledge applications zoo. https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo (August 2021), IAIK, Graz University of Technology
61. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. Cryptology ePrint Archive, Paper 2019/550 (2019), https://eprint.iacr.org/2019/550, https://eprint.iacr.org/2019/550
62. Sivasubramanian, S.: Amazon dynamodb: A seamlessly scalable non-relational database service. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 729–730. SIGMOD '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2213836.2213945, https://doi.org/10.1145/2213836.2213945
63. Szepieniec, A., Lemmens, A., Sauer, J.F., Threadbare, B., Al-Kindi: The tip5 hash function for recursive starks. Cryptology ePrint Archive, Paper 2023/107 (2023), https://eprint.iacr.org/2023/107, https://eprint.iacr.org/2023/107
64. Vujičić, D., Jagodić, D., Randić, S.: Blockchain technology, bitcoin, and ethereum: A brief overview. In: 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH). pp. 1–6 (2018). https://doi.org/10.1109/INFOTEH.2018.8345547
65. Wang, D.: Loopring. https://loopring.org/ (2020), loopring Project Ltd.

66. Winternitz, R.S.: Producing a one-way hash function from des. In: Chaum, D. (ed.) Advances in Cryptology: Proceedings of Crypto 83. pp. 203–207. Springer US, Boston, MA (1984). https://doi.org/10.1007/978-1-4684-4730-9_17, https://doi.org/10.1007/978-1-4684-4730-9{_}17
67. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38