Efficient Symmetric Primitives for Advanced Cryptographic Protocols
(A *Marvellous* Contribution)

Abdelrahaman Aly[1], Tomer Ashur[1], Eli Ben-Sasson[2], Siemen Dhooghe[1], and Alan Szepieniec[1,3]

[1] imec-COSIC, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] StarkWare Industries Ltd
firstname@starkware.co
[3] Nervos Foundation
firstname@nervos.org

Abstract While common symmetric primitives like the AES and SHA3 are optimized for efficient hardware and software implementations, a range of emerging applications using advanced cryptographic protocols such as multi-party computation and zero-knowledge proofs require optimization with respect to a different metric: *arithmetic complexity*. We propose two families of symmetric primitives — *Vision* and *Rescue* — designed specifically to have a compact algebraic description optimizing the advanced cryptographic protocols that employ them. *Vision* operates on a small number of binary field elements and adopts the overall design strategy, and thus the security rationale, of the AES: it is a substitution permutation network whose nonlinear layer composes finite field inversion with an affine transform. To make the same security rationale work for prime fields, the nonlinear layer of *Rescue* consists of a low-degree power map in even steps and its high-degree inverse in odd ones. The algebraic simplicity of the proposed ciphers raises the need for careful cryptanalysis, with a particular focus on algebraic attacks. After providing an elaborate security analysis, we proceed to evaluate the performance of our ciphers with respect to three use cases: the ZK-STARK proof system, proof systems based on Rank-One Constraint Satisfaction (R1CS) systems; and Multi-Party Computation (MPC) with masked operations.

1 Introduction

Block ciphers are a fundamental primitive of modern cryptography. They are used in a host of symmetric-key constructions, *e.g.*, directly as a pseudorandom permutation to encrypt a single block of data; inside a mode of operations to generate a stream cipher for authenticated encryption; or, after some tweaking, in a Merkle-Damgård or sponge construction to generate hash functions. This last example, hash functions, are themselves a fundamental primitive in their

own right for their fitness to approximate a random oracle, and thereby admit a security proof based on this idealization.

While the security of standard block ciphers and hash functions such as the AES, 3DES, SHA2-256, SHA3/Keccak, is well understood and widely agreed on, their design targets an efficient implementation in software and hardware. The design constraints that make these primitives efficient in their niche, are different from the constraints that would make them efficient for use in *advanced cryptographic protocols* such as zero-knowledge proofs, and multi-party computation (MPC). The mismatch in design constraints has prompted a departure from the standardized basic algorithms in favor of new designs, such as LowMC [1] and *MiMC* [2]. We contribute to this collection of primitives.

1.1 Arithmetization and Advanced Cryptographic Protocols

Before we introduce our families of permutations and the design decisions that distinguish them, it is important to understand the needs that they address. To this end, we first survey various application scenarios within the advanced cryptographic protocols considered in this paper, and summarize the constraints induced by them.

The protocols surveyed below use a method that was originally introduced by Razborov [26] in the context of computational complexity and first applied to cryptographic protocols by Lund *et al.* [21]. This method, known as *arithmetization*, characterizes a computation as a sequence of natural arithmetic operations on finite field elements. This characterization translates computational problems — such as determining whether a nondeterministic Turing machine halts in T steps, or whether a Boolean circuit is satisfiable — into algebraic problems involving low-degree multivariate polynomials over a finite field. A subsequent interactive proof system that establishes the consistency of these polynomials simultaneously establishes that the computation was performed correctly. Similarly, the arithmetic properties of finite fields enable the transformation of a computational procedure for one machine — for instance, calculating the value of a function $f(x_1, x_2, x_3)$ — into a procedure to be run jointly by several interactive machines. The practical benefit of this transformation stems from the participants' ability to provide secret inputs x_i, and to obtain the function's corresponding value without revealing any more information about those inputs than is implied by this evaluation. In both cases, the complexity of the derived protocol is determined by that of the arithmetization. Devising cryptographic primitives that have efficient arithmetizations is the core purpose of this paper.

Zero-knowledge proofs. A zero-knowledge (ZK) proof system is a protocol between a prover and a verifier whereby the former convinces the latter that their common input ℓ is a member of a language $\mathcal{L} \subset \{0,1\}^*$. The proof system is complete and sound with soundness error ϵ if it guarantees that the verifier accepts (outputs 1) when $\ell \in \mathcal{L}$ and rejects with probability $\geq 1 - \epsilon$ when $\ell \notin \mathcal{L}$. When this soundness guarantee holds only against computationally bounded

provers we call it an *argument* system. The proof system is zero-knowledge if the transcript is independent of the membership or non-membership relation.[4] We are concerned here with languages $\mathcal{L}$ that capture generic computations in different models of computation.

Scalable, transparent arguments of knowledge. Let $\mathcal{L}$ be a language decidable in nondeterministic time $T(n)$, like the NEXP-complete bounded halting problem,

$$\mathcal{L}_H = \{(M, T) \mid M \text{ is a nondeterministic machine that halts within } T \text{ cycles.}\}$$

Following [7], we say that a ZK proof system for $\mathcal{L}$ is

- *scalable* if two conditions are satisfied simultaneously for all instances $\ell, |\ell| = n$: (i) proving time scales quasi-linearly, like $\mathsf{poly}(n) + T(n) \cdot \mathsf{poly}\log T(n)$, and (ii) verification time scales like $\mathsf{poly}(n) + \mathsf{poly}\log T(n)$.
- *transparent* if all verifier messages are public coins. These systems require no trusted setup phase.
- *argument of knowledge* if there exists an extractor that efficiently recovers a witness of membership of ℓ in $\mathcal{L}$ by interacting with a prover whose messages have a sufficiently high probability of acceptance by the verifier.

Argument systems that possess all of the properties above are referred to as ZK-STARKs, and have been recently implemented in practice [7], following theoretical constructions [8, 9] (cf. [6] for a prior, non-ZK, STARK).

To reap the benefits of a scalable proof system, it is important to encode computations succinctly, and one natural way to achieve this is via an Algebraic Intermediate Representation (AIR), as suggested in [7]. Both Turing machines and Random Access Memory (RAM) machines can be represented succinctly using AIRs that we describe briefly now, and more formally in the appendix. [5]

An Algebraic Execution Trace (AET) is similar to an execution trace of a computation. It is an array with t rows (one row per time step) and w columns (one column per register). The *size* of the AET is $\mathsf{t} \cdot \mathsf{w}$. The main property distinguishing an AET from a standard execution trace is that each entry of the array is an element of a finite field $\mathbb{F}_q$. The transition function of the computation is now described by an Algebraic Intermediate Representation (AIR). An AIR is a set $\mathcal{P}$ of polynomials over $2\mathsf{w}$ variables $\boldsymbol{X} = (X_1, \ldots, X_{\mathsf{w}}), \boldsymbol{X'} = (X'_1, \ldots, X'_{\mathsf{w}})$, representing, respectively, the current and next state of the computation, such that a transition from state $\boldsymbol{s} = (s_1, \ldots, s_{\mathsf{w}}) \in \mathbb{F}_q^{\mathsf{w}}$ to state $\boldsymbol{s'} = (s'_1, \ldots, s'_{\mathsf{w}}) \in \mathbb{F}_q^{\mathsf{w}}$ is valid iff all polynomials in $\mathcal{P}$ evaluate to 0 when the values $\boldsymbol{s}, \boldsymbol{s'}$ are assigned to the variables $\boldsymbol{X}, \boldsymbol{X'}$, respectively. (See appendix B for an example.)

To maximize the efficiency of ZK-STARKs, we wish to minimize the three main parameters of the AIR: the computation time t, the state width w and the maximal degree d of an AIR constraint (polynomial) in $\mathcal{P}$.

[4] Specifically, if authentic transcripts are indistinguishable from transcripts that can be generated even when $\ell \notin \mathcal{L}$ by not respecting the correct order of messages.

[5] Dealing with random access memory requires a variant of an AIR — a Permuted AIR (PAIR), but all computations discussed later on in this paper can be done with a constant number of registers.

Circuit model. Numerous ZK proof systems operate in the model of arithmetic circuits, meaning that the language $\mathcal{L}$ is that of satisfiable arithmetic circuits (ones whose output is 0). Succinct computations can be "unrolled" into arithmetic circuits, and several compilers exist that achieve this, *e.g.*, [10,25,28]. Such circuits are specified by directed acyclic graphs with labeled nodes and edges. The edges, or *wires*, have a value taken from some ring; the nodes, or *gates*, perform some operation from that ring on the values contained by its input wires and assign the corresponding output value to its output wires. An assignment to the wires is valid if and only if for every gate, the value on the output wires matches that gate's operation and the values on its input wires. In the context of zero-knowledge proofs, the prover generally proves *knowledge* of an assignment to the input wires of a circuit computing a one-way function, meaning that the corresponding output matches a given public output. Alternatively, the prover can prove *satisfiability* — that there exists a corresponding input — which makes sense in the context where it is also possible for no such input to exist.

Recent years have seen a concentration of effort towards Quadratic Arithmetic Programs (QAPs) [19] and rank-one constraint satisfaction (R1CS) systems [10] for encoding circuits and wire assignments in an algebraically useful way. The circuit is represented as a list of triples $((\boldsymbol{a_i}, \boldsymbol{b_i}, \boldsymbol{c_i}))_i$. A vector $\boldsymbol{s}$ of assignments to all wires is valid iff $\forall i \,.\, (\boldsymbol{a_i}^\mathsf{T}\boldsymbol{s}) \cdot (\boldsymbol{b_i}^\mathsf{T}\boldsymbol{s}) = \boldsymbol{c_i}^\mathsf{T}\boldsymbol{s}$. R1CS systems can be defined over any ring; when this ring is $\mathbb{Z}/p\mathbb{Z}$, *i.e.*, the *field* of integers modulo some prime p, the R1CS instance captures exactly an intermediate step of the ZK-SNARK family of proof systems [19]. Additional transparent systems such as Ligero [3], BulletProofs [13] and Aurora [11] also accept R1CS over different fields as their input. For the purpose of efficient R1CS-style proofs, the degree of the constraints describing a cipher is as important as their number: any algebraic constraint of degree higher than two must first be translated into multiple constraints of degree two, and the complexity parameter we seek to minimize is the number of R1CS gates (or constraints) needed to specify the computation.

Multi-party computation (MPC). A multi-party computation is the joint evaluation of a function in individually known but globally secret inputs. In recent years, MPC protocols have converged to a linearly homomorphic secret sharing scheme whereby each participant is given a share of each secret value such that locally adding shares of different secrets generates the shares of the secrets' sum. We use the bracket notation $[\cdot]$ to denote shared secrets.

To compute products, the MPC protocol consumes one pre-generated multiplication triple, which is a triple of shared and secret values $([a], [b], [c])$ such that $ab = c$. Since additions are essentially free and multiplication requires the consumption of multiplication triples, the number of such triples required to perform a computation is a good first estimate of the complexity of an MPC protocol.

However, while one multiplication requires one round of communication, in many cases it is possible to batch many multiplications into a single round. Moreover, some communication rounds can be executed in an offline pre-processing

phase before receiving the input to the computation. These offline rounds are cheaper than the online rounds, as the former does not affect the protocol's latency and the latter completely determines it. To assess the MPC-friendliness of a cipher one must therefore take three metrics into account: *number of multiplications*; *number of offline communication rounds*; and *the number of online communication rounds*.

An important family of techniques that have a relatively low multiplication count, offline and online complexity is masked operations such as the technique suggested by Damgård *et al.* [17]. The protocol raises a shared secret to a large power while offloading the bulk of the computation to the offline phase. Suppose for instance that the protocol wishes to compute $[a^e]$ for some exponent e, given only the shared secret $[a]$. The protocol generates a random nonzero blinding factor $[r]$ and computes $[r^{-e}]$ using $\log_2 e$ multiplications in the offline phase. In the online phase they multiply $[a]$ with $[r]$, open $[ar]$, and then locally raise this known number to the power e. The result of this exponentiation is then multiplied with $[r^{-e}]$ giving $(ar)^e[r^{-e}] = [a^e r^e r^{-e}] = [a^e]$. A similar procedure enables the computation of inverses with only a handful of multiplications [4]. We extend this range of techniques in two ways. First, we adapt the technique of Damgård *et al.* for exponents of the form $1/\alpha$; while the online complexity is the same, our technique reduces the offline complexity by exploiting the concise representation of α. Second, we introduce a new technique to efficiently evaluate the compositional inverse of sparse linearized polynomials. This novel technique is a contribution of independent interest. We cover these masked operation techniques in more detail as part of our efficiency evaluation. The key observation is that *some* polynomials with large powers *can* be efficiently computed over MPC — *even when counting the offline phase.*

1.2 A *Marvellous* Contribution

This paper presents the *Marvellous* universe of cryptographic primitives optimized for efficiency in advanced cryptographic protocols such as MPC and zero-knowledge proof systems based on either model of computation. The universe consists of two families, *Vision*[6] and *Rescue*[7], operating over fields of even and odd order, respectively. Both families manipulate a state of $m > 1$ elements for N_b rounds. At a constant level of security, m and N_b are inversely proportional, meaning that m represents a parameter that can be tweaked in order to favor a lower multiplication depth in exchange for a higher multiplication count.

Vision follows the design rationale and security argument of the Rijndael family of block ciphers. Unlike Rijndael, which operates in each round on 16 elements living in $\mathbb{F}_{2^8}$ (*i.e.*, bytes), *Vision* operates in each round on $m > 1$ elements in $\mathbb{F}_{2^{n/m}}$ where n and m are chosen depending on the task at hand. The

[6] In the Marvel universe, Vision is an android created by Tony Stark *et al.* and powered by binary extension fields.

[7] In the Marvel comics, Pepper Potts, Chief Executive Officer of Stark Industries, is a prime character and an occasional superhero under the name Rescue.

inversion takes place over $\mathbb{F}_{2^{n/m}}$ and the affine layer is constructed specifically to have a low algebraic complexity — thereby enabling efficient prover or multi-party computations.

Rescue also operates on $m > 1$ field elements but this time over a medium or large prime field $\mathbb{F}_p$. Translating the design rationale from Rijndael and Vision to work over prime fields presents a couple of difficulties owing to the non-existence of higher-degree affine transforms. Our solution is to merge the inversion and affine steps into one step: raising each state element to the power α or $1/\alpha$, where α is the smallest prime for which the power map is bijective. This merger recycles the security argument as this α-power map simultaneously is highly nonlinear, has high rational degree, and delocalizes differences.

By fixing the secret key, both block ciphers become permutations. By using this permutation as the main building block in a sponge construction [12], we obtain extendable output functions and, if the output length is fixed, hash functions. We note that the sponge mode can also be used to turn the permutations into stream ciphers. However, exploring this option is beyond the scope of this paper.

1.3 Structure of this Paper

The structure of this paper is as follows: in Section 2 we consider previous works, such as the cryptographic primitives Rijndael and *MiMC*, and the sponge framework. In Sections 3–4 we present our respective designs *Vision* and *Rescue*. Then, in Sections 5–8 we evaluate the efficiency of our constructions in advanced cryptographic protocols and compare them to *MiMC*: Section 5 evaluates them in AIR; Section 6 evaluates them in R1CS; Section 7 evaluates them in masked MPC protocols. A comparison for the evaluations is given in Section 8. Finally, Section 9 concludes the body of this paper.

In Appendix A the interested reader will find the experimental Gröbner bases analysis we used for determining the number of rounds. In Appendices B–C we recall the relevant definitions related to STARKs, along with a simple motivating example. Appendix D features pseudo-code of the masking algorithms we used in Section 7. Finally, in Appendix E we present a straightforward square-and-multiply implementation of *Vision*, *Rescue*, and *MiMC*.

2 Related Work

In this section we recall relevant previous works.

2.1 The Sponge Framework

A sponge construction generates a hash function from an underlying permutation by iteratively applying it to a large state. The state consists of $b = r + c$ bits, where r and c are called the *rate* and the *capacity* of the sponge, respectively. The construction hashes a variable length-input into an arbitrary-length output

by absorbing r bits of the input at every iteration. We refer the interested reader to [12] for the full details and intricacies of the sponge construction, the padding schemes that are compatible with it, and its security claims.

2.2 Rijndael-128

Since the first block cipher we introduce, namely *Vision*, is obtained by generalizing the Rijndael block cipher, we start with a brief description of the latter. The Rijndael-128 cipher, better known as AES-128, consists of five building blocks: AddRoundKey, SubBytes, MixColumns, ShiftRows and ExpandKey (which in turn includes: SubWords, AddWords, RotWords and AddConstants). For our new constructions, we focus mainly on changing the S-Boxes (*i.e.*, SubBytes and SubWords) which we thus recall in more detail. For more information on each building block and the security they give, we refer the reader to [16].

Rijndael S-Box For Rijndael, each S-Box works over one byte and consists of the sequential execution of two functions $S\text{-}Box(z) = g(f(z))$. The first function f is defined as the adapted multiplicative inverse function over $\mathbb{F}_{2^8}$ where zero is defined to be mapped to zero,

$$f : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8} : x \mapsto x^{254} \ .$$

This function protects against differential and linear attacks, and allows for the wide trail design strategy together with the MixColumns and ShiftRows diffusion functions. The second function g in the SubBytes step is the affine transformation

$$g : \mathbb{F}_2^8 \to \mathbb{F}_2^8 : x \mapsto Mx + b \ ,$$

with $M \in \mathbb{F}_2^{8\times 8}$ and $b \in \mathbb{F}_2^8$. The main property of this transformation is to make the polynomial representation of the S-Box over $\mathbb{F}_{2^8}$ more complex and thus to increase the resistance of the cipher against algebraic attacks. Note that while the affine transformation works over $\mathbb{F}_2$, the entire S-Box can be represented as the following polynomial over $\mathbb{F}_{2^8}$,

$$\begin{aligned} S\text{-}Box(z) = \text{0x05} \cdot z^{254} + \text{0x09} \cdot z^{253} + \text{0xF9} \cdot z^{251} + \text{0x25} \cdot z^{247} + \text{0xF4} \cdot z^{239} \\ + \text{0x01} \cdot z^{223} + \text{0xB5} \cdot z^{191} + \text{0x8F} \cdot z^{127} + \text{0x63} \ . \end{aligned}$$

2.3 The Wide Trail Strategy

From [15, Section 3.1]: *"The wide trail design strategy was introduced as a means to guarantee low maximum probability of multiple-round differential trails and low maximum correlation of multiple-round linear trails."* This strategy is used to parameterize a cipher's resistance against differential and linear cryptanalytic attacks. In the design of Rijndael [16], Daemen and Rijmen look at four rounds of Rijndael-128. By using properties of the linear layers, they prove that for any input there will always be at least 25 active S-Boxes.

Next, they argue cryptanalytic properties of an S-Box considering differential and linear cryptanalysis, namely the difference propagation probability and the maximum absolute correlation. The difference propagation probability δ of an n-bit Boolean function f is defined as

$$\delta = 2^{-n} \max_{i,j} \ |\{x \mid f(x) \oplus f(x \oplus i) = j\}| \ .$$

The maximum absolute correlation between any pair of linear combinations of n input bits and n output bits λ over f is defined as

$$\lambda = \max_{\alpha,\beta \in \mathbb{F}_2^n} \left(2 \Pr_{a \in \mathbb{F}_{2^n}} [\alpha a \oplus \beta f(a) = 0] - 1 \right)$$

where α and β are the input and output masks, respectively (for the complete argument and definitions we refer the reader to [16]).

The cryptanalytic properties of the inversion function in the S-Box of Rijndael were obtained by Nyberg [24]. For $\mathbb{F}_{2^8}$ we have $\delta = 2^{-6}$ and $\lambda = 2^{-3}$.

As there are at least 25 active S-Boxes in four rounds and every S-Box has a difference propagation probability of at most $\delta = 2^{-6}$ and a maximum absolute correlation $|\lambda| = 2^{-3}$, a four round differential trail will have a maximal probability of 2^{-150} and a maximal absolute correlation of 2^{-75}. This means that an eight round trail has a maximal probability of 2^{-300} and maximum absolute correlation 2^{-150} which the designers deem sufficient to resist differential and linear attacks.

2.4 *MiMC*

MiMC refers to a pair of block ciphers developed by Albrecht *et al.* [2] and designed to work over finite fields $\mathbb{F}_q$ where q is a prime or a prime power of 2. The first variant, *MiMC-q/q*, consists of a state of one field element in $\mathbb{F}_q$ which is raised to the power 3 in each round followed by the addition of a random constant C_i and the same key K is injected in every round. The construction is depicted in Figure 1. For a 128-bit state, 82 rounds are sufficient to offer full security.[8]

The second variant of *MiMC* uses a Feistel network and is denoted by *MiMC-2p/p*, where p is once again a prime or a prime power of 2. The construction is depicted in Figure 2. The state consists of two $\mathbb{F}_p$ elements, denoted x_L and x_R. The ℓth round function is

$$x_L || x_R \leftarrow x_R + (x_L + K + C_\ell)^3 || x_L \ ,$$

where K is the key and c_ℓ the ℓth round constant. For a 128-bit state, 164 rounds are enough to offer full security.

[8] In case the attacker has restricted plaintext/ciphertext pairs or limited memory, the number of rounds can be reduced. For a fair comparison, we consider only the general case.

Additionally, the authors propose two hash functions: *MiMCHash-q/q* and *MiMCHash-2q/q*. Both are sponge constructions using the permutation obtained by fixing the key of the respective block cipher to the all-zero string. The absorption and squeezing of input and output in the case of *MiMCHash-q/q* are not native operations to the working field and thus require complex arithmetic to achieve. By contrast, *MiMCHash-2q/q* is much better suited to arithmetization and thus what we compare our algorithms (in hash mode) against.

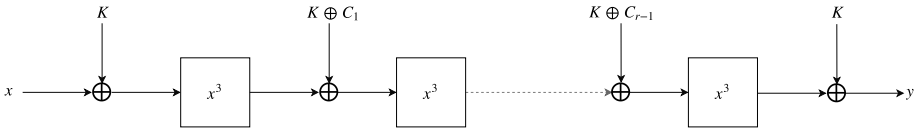


Figure 1: The *MiMC-q/q* block cipher as shown in [2]

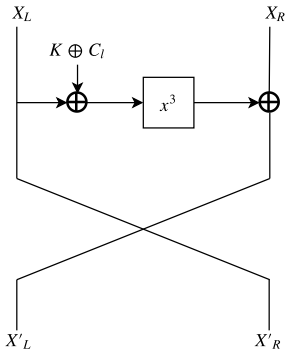


Figure 2: The *MiMC-2p/p* block cipher as shown in [2]

3 *Vision*

We now take the Rijndael cipher and generalize it to $\mathbb{F}_{2^{n/m}}^m$ where we aim for n bits of security working with an n-bit state and an n-bit key. Incidentally, this greatly improves (compared to Rijndael) the algebraic efficiency of the design such as the number of multiplications.

In Section 3.1 we describe the round function of *Vision* and its key schedule. Then, in Section 3.2 we discuss the security of the algorithm, and in Section 3.3 how to choose the number of rounds.

3.1 Round Function

In the design of *Vision* we make a distinction between a *step* and a *round*. A step in *Vision* corresponds to a round in Rijndael and includes a nonlinear operation, an affine layer, an MDS matrix, and a key injection. A round of *Vision* consists of two steps.

The most significant change in the new construction is that it works with larger S-Boxes hence reducing their overall number. We bundle together the state into $m > 1$ elements, thus creating an S-box that is a nonlinear function over $2^{n/m}$ bits rather than over bytes. We adapt the inversion function for this field. By Fermat's little theorem we get

$$f : \mathbb{F}_{2^{n/m}} \to \mathbb{F}_{2^{n/m}} : x \mapsto x^{2^{n/m}-2} \ ,$$

or in rational form

$$f(x) = \begin{cases} 1/x, & \text{if } x \neq 0\,. \\ 0, & \text{otherwise}\,. \end{cases}$$

Similar to the S-Box of Rijndael, the multiplicative inverse is followed by an affine polynomial. Recall that an $\mathbb{F}_2$-linearized polynomial is of the form

$$L(x) = \sum_{i=0}^{n/m-1} b_i x^{2^i} \in \mathbb{F}_{2^{n/m}}[x] \quad \text{with all} \quad b_i \in \mathbb{F}_{2^{n/m}} \ .$$

Such a polynomial is a permutation over $\mathbb{F}_{2^{n/m}}$ if and only if $L(x)$ only has the root 0 in $\mathbb{F}_{2^{n/m}}$. Finally we add a constant to this linearized polynomial, creating an $\mathbb{F}_2$-affine polynomial,

$$B(x) = b_{-1} + \sum_{i=0}^{n/m-1} b_i x^{2^i} \in \mathbb{F}_{2^{n/m}}[x] \ .$$

In the nonlinear layer, each of the m elements goes through an inversion operation, followed by an affine polynomial: either $B^{-1}(x)$ if this is the first step in the round, or $B(x)$ if it is the second one. The coefficients b_i of $B(x)$ are chosen such that they are guaranteed not to lie in any subfields of $\mathbb{F}_{2^{n/m}}$ in order to thwart subfield attacks. The polynomial degree of $B(x)$ is four for all n, m, but as every round involves one evaluation of $B^{-1}(x)$ and of $B(x)$, the algebraic degree of a full round in either direction (*i.e.*, encryption or decryption) is large. Nevertheless, the evaluations of both $B(x)$ and $B^{-1}(x)$ remain efficient for prover and multi-party computations precisely because of this relatively low degree.

Finally, we replace the ShiftRows and the MixColumns operations by a single MDS matrix M generated from a $[2m, m, m+1]_{2^{n/m}}$ MDS code. We denote $M[i, j]$ the (i, j)-entry of M. A schematic description of a single round (two steps) of *Vision* is depicted in Figure 3 and a full description of the cipher is listed in Algorithm 1.

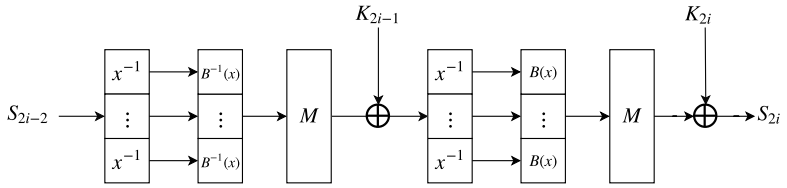


Figure 3: A single round (two steps) of *Vision*

Algorithm 1: *Vision*

Input: Plaintext P, round keys K_r for $0 \leq r \leq 2N_b$
Output: *Vision* (K, P)

$State_0 = P + K_0$
for $r = 1$ **to** N_b **do**
  for $i = 1$ **to** m **do**
    $Inter_r[i] = (State_{r-1}[i])^{-1}$
    $Inter_r[i] = B^{-1}(Inter_r[i])$
    $State_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[j] + K_{2r-1}[i]$
  end
  for $i = 1$ **to** m **do**
    $Inter_r[i] = (State_r[i])^{-1}$
    $Inter_r[i] = B(Inter_r[i])$
    $State_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[j] + K_{2r}[i]$
  end
end
return $State_{N_b}$

Key schedule. Similar to Whirlpool [5] and in the interest of simplicity, we propose to reuse the round function of *Vision* for the key schedule as well. We take a step of the key schedule to be equal to a step of the cipher proper, replacing the key addition with an addition of predetermined constants. We set $K_0 = K$ and run the key schedule. The output of step ℓ of the key schedule is then used as the subkey of step ℓ in the cipher proper. A schematic view of the key schedule is depicted in Figure 4

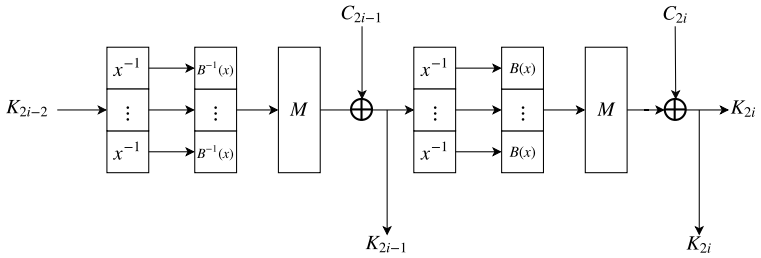


Figure 4: The key schedule of *Vision*

Given the first round constant, all subsequent constants are obtained by applying an $\mathbb{F}_{2^{n/m}}$ affine transformation to the previous one. We use SHAKE256 to expand a short seed into enough randomness from which one samples (with rejection as necessary) the initial round constant; the coefficients of this $\mathbb{F}_{2^{n/m}}$ affine transformation; and the coefficients of the $\mathbb{F}_2$ affine transformation B. These constants are generated deterministically using the code provided in [27].

Hashing. By fixing the key to the all zeros string, we transform *Vision* into a permutation thereby enabling its use in a sponge construction. The input is padded and split into blocks of $r_{2^{n/m}}$ elements in $\mathbb{F}_{2^{n/m}}$. The remaining $m - r_{2^{n/m}}$ elements of the state constitute the capacity and determine the security of the sponge. After absorbing all the message blocks, $r_{2^{n/m}}$ elements can be squeezed out in every subsequent iteration. *Vision* is agnostic to the specific sponge-construction used.

3.2 Cryptanalytic Strength

In this section we explain the security of *Vision* and how it resists certain attacks.

The Wide Trail Strategy In order to argue the security of *Vision*, we follow the same line of reasoning as was done for Rijndael and apply the wide trail strategy to our construction. From Nyberg [24] we take the differential and linear properties of the inversion function over arbitrary binary fields. For the field $\mathbb{F}_{2^{n/m}}$ we have $\delta = 2^{-n/m+2}$ and $|\lambda| = 2^{-\lceil n/2m \rceil + 1}$. Since our construction

uses m S-Boxes in every step which are mixed with an MDS matrix, we have at least $m+1$ active S-Boxes after a single round consisting of two steps. We require that $n/m \geq 4$. Thus a four round trail has a maximal differential probability of

$$2^{4(m+1)(-n/m+2)} < 2^{-2n} \ ,$$

and maximum absolute correlation

$$2^{-4(m+1)\lceil n/2m \rceil + 4(m+1)} < 2^{-n} \ ,$$

which is sufficient to resist potential differential and linear attacks given that a large enough security margin was taken.

Polynomial Expressions Jakobsen and Knudsen introduced in [20] the interpolation attack. Here the attacker constructs polynomials using input/output pairs of the cipher. Due to the complexity of calculating GCD's or Lagrange interpolation being linear in the degree of the polynomial, one needs the polynomial representations of the cipher to have a high degree to avoid this attack. Recall that the inversion composed with an affine mapping over $\mathbb{F}_{2^{n/m}}$ can be expressed as the polynomial

$$b_{-1} + \sum_{i=0}^{n/m-1} b_i x^{2^{n/m}-2^i-1}, \text{ for } b_i \in \mathbb{F}_{2^{n/m}} \ .$$

For our construction the polynomial expression of one round is of degree close to the maximum $(2^{n/m}-1)$. Due to the inversion function after two steps, any polynomial expression would also be dense.

Rational Expressions The rational degree is defined as the maximum of the degree of the numerator and of the denominator of the cipher's fractional representation. Denote the rational expression of an S-Box as $p(x)/q(x)$ such that the degree d of this ratio of polynomials is minimal. We see that N_b iterations of such an S-Box then would create a rational polynomial of maximal degree d^{N_b}. The S-Box consisting of the inversion function with an affine layer looks as follows,

$$S\text{-}Box(x) = w_{-1} + \sum_{i=0}^{n/m-1} \frac{w_i}{x^{2^i}} \ ,$$

where each w_i is the coefficient of the polynomial representation of the affine layer. We see that the degree of the rational representation of the S-Box is equal to 2^d for the maximal d such that $w_d \neq 0$. Thus the maximal rational degree of the cipher using N_b iterations is $2^{N_b d}$, where the maximal degree is equal to $2^{n/m}-1$. Due to the combination between the affine layer and the inversion function, the rational expression is dense. Thus, our dense expression of degree $2^{N_b d}$ should have around $2^{N_b d}$ rational terms after N_b iterations. Since $B^{-1}(x)$

is of full degree, two rounds of the cipher are sufficient to create a complex rational expression between the plaintext and the ciphertext. We also note that attacks using the rational expression of the cipher can have meet-in-the-middle attack variants, such as with the interpolation attack [20], thus we consider three rounds of the cipher to be sufficient in order to resist these attack variants.

Invariant Subfield Attacks Finally, we consider attacks which make use of an invariant subfield. To recall, for $\mathbb{F}_{2^{n/m}}$, any field $\mathbb{F}_{2^s}$ where s is a divisor of n/m is a subfield. An adversary might be able to attack the cipher by making it work over one of the subfields. This would involve the adversary inputting a value of a subfield and receiving an output which is again in the same subfield. When looking at the polynomial representation of a round in *Vision*. We require that the affine polynomial has coefficients which do not lie in any subfield of $\mathbb{F}_{2^{n/m}}$ thus frustrating this attack.

Attacks Using Gröbner Bases A Gröbner basis attack amounts to solving a linear system whose coefficient matrix is the extended Macaulay matrix of a list of polynomials, in which every row represents one polynomial equation and every column represents one monomial. Starting with an original list of polynomials that uniquely defines the solution, this list is extended in two ways: one, by multiplying polynomials with monomials; and two, by adjoining the newly derived possibly linearly independent polynomial equations to the working list. Eventually, the procedure will add rows faster than columns and at some point solving the linearized system using fast linear algebra techniques produces a solution to all the polynomial equations, the derived ones as well as the original ones.

The attack scenario we consider here is a preimage attack when *Vision* with a fixed key is used as a permutation inside a sponge function. In our analysis we observed that the degree of regularity grows very slowly as a function of the number of rounds whenever $m = 1$. As a result, we recommend instantiating families of the *Marvellous* universe with $m > 1$. Consequently a sponge construction represents a native approach.

To perform this preimage attack on *Vision*, one compiles a list of polynomial equations associated with one sponge permutation absorbing one unknown data block and squeezing out one known block. Absorbing more than one data block just increases the number of variables but has no effect otherwise; squeezing out more than one data block increases the number of equations and thereby decreases the attack complexity. Nevertheless we focus on one output block as it captures pertinent use cases.

The following system of equations encodes one full round of *Vision*. Here $S_{2i-1} \in \mathbb{F}_{2^{n/m}}^m$ is the intermediate state in the middle of Fig. 3, $\odot$ represents the component-wise product, the evaluation of B is component-wise as well, and 0 and 1 represent the vectors of m components set to 0 and 1, respectively. The superscript -1 refers to component-wise inversion for S_{2i-1} and to matrix

inversion for M.

$$\left\{ \begin{array}{l} S_{2i-2} \odot B\big(M^{-1}(S_{2i-1} - K_{2i-1})\big) - 1 = 0 \\ S_{2i-1}^4 \odot B\big(S_{2i-1}^{-1}\big) - S_{2i-1}^4 \odot M^{-1}(S_{2i} - K_{2i}) = 0 \end{array} \right\}$$

Note that left hand side of the second equation is a polynomial in S_{2i-1} as the negative powers are cancelled by the factor S_{2i-1}^4 and as the degree of the affine polynomial B is 4.

In total, this system represents $2m$ polynomial equations of degree 5 in $2m$ extra variables per round. The state at the end gives one equation, as one element is squeezed out; and $m-1$ variables, as the value of the one that is squeezed out is known. The state at the start gives one variable representing the unknown absorbed data block and $m-1$ equations equating parts of the state to zero. So in total there are $2mN_b$ variables and as many equations, with N_b the number of rounds and all polynomials are of degree 5. Assuming the system is regular we find that the Hilbert Series $HS(z) = \frac{\prod_{i=0}^{2mN_b-1}(1-z^5)}{(1-z)^{2mN_b}}$ is a polynomial of degree $8mN_b$, thereby indicating a degree of regularity of $d_{reg} = 8mN_b + 1$.[9] Polynomials in $2mN_b$ variables of this degree have $\binom{2mN_b+d_{reg}}{2mN_b}$ monomials, and squaring this number gives a lower bound on the complexity of solving the linearized system and hence of the overall Gröbner basis attack.

3.3 Choosing the Number of Rounds

In this section we analyze how many rounds are required for the cipher to be secure. Evidently, this part requires special attention as it affects both the security and efficiency of *Vision*. The number of rounds is determined by the desired security which in turn is determined by the feasibility of the attacks described above. The quality of such attacks is quantified by the wide trail strategy, by the maximal degree and number of terms of the polynomial and rational expressions of the cipher, and by the complexity of possible Gröbner basis attacks.

As the other attacks become infeasible after a small number of rounds, the bottleneck seems to be the complexity of a Gröbner basis attack. From our analysis and subsequent least-squares fit we find that the base-2 logarithm of the complexity of such an attack is lower-bounded by $5.5mN_b$. Accounting for a factor 2 security margin, we recommend $2\lceil n/5.5m \rceil$ rounds, with a minimum of 10 rounds, for ciphers operating on a state of m elements and targeting an n-bit security level.

The Gröbner basis attack scenario we considered here, and according to which we determined the number of rounds, was tailored for using *Vision* in sponge-based hashing mode as described in Section 2.1. When used as a block cipher, the resistance to Gröbner basis attacks would only increase as a result of adding more variables through the key schedule. We therefore give $2\lceil n/5.5m \rceil$ as a highly conservative bound and hope that third-party analysis reaches the conclusion that a smaller number of rounds is sufficient.

[9] Experiments we conducted for small m and N_b empirically validate the assertion that the system behaves like regular ones do. These results are shown in Appendix A.

4 *Rescue*

The second family of algorithms in the *Marvellous* universe is *Rescue*. *Rescue* is similar to *Vision*, but this time operating on elements of $\mathbb{F}_p$ rather than on elements of $\mathbb{F}_{2^{n/m}}$.

However, the transition from *Vision* to *Rescue* is not straightforward. Since higher-degree affine polynomials do not exist for $\mathbb{F}_p$ we need to find another way to increase the algebraic degree in order to attain the same security level. One way to do this is to replace the inverse function with another power mapping α. This is in line with *MiMC* [2] which uses $\alpha = 3$. Unlike the case of *MiMC*, in which security is obtained by employing a relatively large number of rounds, we increase the algebraic degree by alternating between α and $1/\alpha$ in consecutive steps. We note that both α and $1/\alpha$ are of low-degree when computed in their "forward" direction and of high-degree when computed in the other direction. This property makes them suitable for our needs.

In Section 4.1 we describe *Rescue* and its key schedule. *Rescue* works with a power mapping α and in Section 4.2 we discuss proper choices for this α. In Section 4.3 we argue the security of the algorithm, and in Section 4.4 how to choose the number of rounds.

4.1 The Round Function

The round function of *Rescue* is depicted in Figure 5. The cipher works over elements of the prime field $\mathbb{F}_p$ with α the smallest prime such that $\gcd(p-1, \alpha) = 1$. The cipher provides a security level of $\log_2(p^m)$ bits, *i.e.*, m times the bit size of the prime modulus.

Rescue over $\mathbb{F}_p^m$ carries m field element as its state which goes through a nonlinear operation, an MDS matrix M, and a key injection in each step. Here, the power map is $x^{1/\alpha}$ if this is the first step in the round, and x^α if it is the second. This nonlinear step is followed by an MDS matrix and a round key addition over $\mathbb{F}_p^m$.

We focus our attention on the inverse power map which is the function $f : \mathbb{F}_p \to \mathbb{F}_p$ such that $\forall x \in \mathbb{F}_p : f(x^\alpha) = x$ or $f(x)^\alpha = x$. We note that this power map exists since we required that $\gcd(p-1, \alpha) = 1$. More specifically, $1/\alpha \cdot \alpha \equiv 1 \bmod p - 1$.

A schematic description of the round function (two steps) of *Rescue* can be found in Figure 5 and its algorithm is listed Algorithm 2. Note that here, similar to *Vision*, both steps are efficient for prover and multi-party computations owing to the low degree of x^α.

Key schedule. Similar to *Vision* we propose to use the round function for the key schedule. We replace the key addition with a constant addition, and use the output of a step in the key schedule as the subkey for the respective step in the cipher. The initial input to the key schedule is set as $K_0 = K$ and the key schedule is clocked to extract a single round key in every step. A schematic view of the key schedule is depicted in Figure 6. Again, the code provided in [27] deterministically generates these constants.

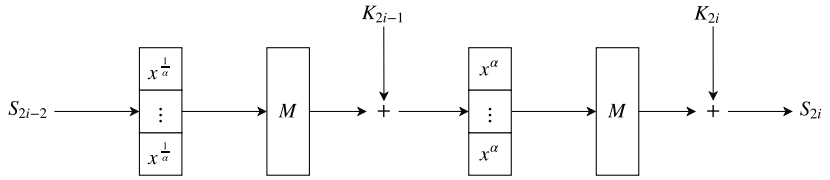


Figure 5: One round (two steps) of *Rescue*.

Algorithm 2: *Rescue*

Input: Plaintext P, round keys K_r for $0 \leq r \leq 2N_b$
Output: *Rescue* (K, P)

$State_0 = P + K_0$
for $r = 1$ **to** N_b **do**
  for $i = 1$ **to** m **do**
    $Inter_r[i] = (State_{r-1}[i])^{1/\alpha}$
    $State_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[j] + K_{2r-1}[i]$
  end
  for $i = 1$ **to** m **do**
    $Inter_r[i] = (State_r[i])^{\alpha}$
    $State_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[i] + K_{2r}[i]$
  end
end
return $State_{N_b}$

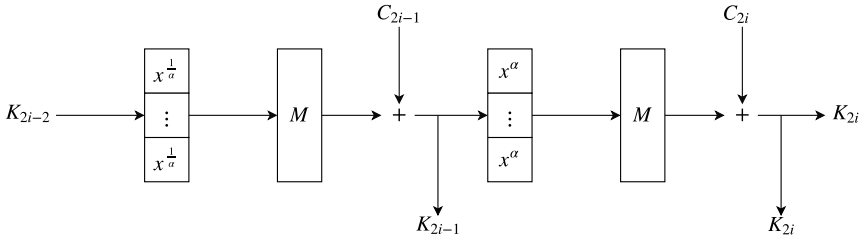


Figure 6: The key schedule of *Rescue*

Hashing. Similar to *Vision*, by fixing the key to the all zeros string, we transform *Rescue* into a permutation thereby enabling its use in a sponge construction. The input is padded and split into blocks of r_p field elements in $\mathbb{F}_p$. The remaining $m-r_p$ elements of the state constitute the capacity and determine the security of the sponge. After absorbing all the message blocks, r_p elements can be squeezed out in every subsequent iteration. *Rescue*, like *Vision*, is agnostic to the specific sponge-construction used.

4.2 Choosing the Power Mapping

For most fields, $\alpha = 3$ suffices, and when possible we recommend to choose the field such that $\alpha = 3$ is viable. However, in some cases the field is determined by the intended application and cannot be chosen freely. For example, the $2^{255} - 19$ field for elliptic curve cryptography does not have $\gcd(p-1, 3) = 1$ making $\alpha = 3$ unsuitable for this case. Instead, users interested in using this field can choose $\alpha = 5$ because $\gcd(p-1, 5) = 1$.

4.3 Cryptanalytic Strength

We now discuss the security of *Rescue*.

Wide Trail Strategy over $\mathbb{F}_p^m$ We look at the difference propagation probability of the function x^α. In other words, we are interested in solutions a, b such that

$$(x+a)^\alpha - x^\alpha - b = 0 \ ,$$

Since this equation is of degree $\alpha - 1$ over x it follows that the α power function is $(\alpha - 1)$-uniform giving the map a difference propagation probability of at most $\delta = 2^{-\log_2(p)+\log_2(\alpha-1)}$ where the differences are taken over $\mathbb{F}_p$. From the requirement $\gcd(p-1, \alpha) = 1$, we know that the α power function is a permutation, thus its inverse exists and has the same differential uniformity. In short, we find that $x^{1/\alpha}$ is also $(\alpha - 1)$-uniform.

From the use of an MDS matrix as diffusion layer between subsequent steps, we know that every two steps consist of minimally $m + 1$ active S-Boxes. From the stochastic equivalence assumption, we find that for four rounds, or eight steps, of *Rescue*, a differential trail has a maximal differential probability of

$$2^{-4(m+1)\log_2(p)+4\log_2(\alpha-1)(m+1)} \ .$$

We require that $\log_2(p) \geq 2\log_2(\alpha - 1)$, thus

$$2^{-4(m+1)\log_2(p)+4\log_2(\alpha-1)(m+1)} \leq 2^{-2s} \ ,$$

with $s = m\log_2(p)$ the bit-level security. This bound is sufficient to argue the algorithm's security against differential attacks.

Linear cryptanalysis. Arguing the resistance of *Rescue* against linear cryptanalysis is somewhat more difficult. Linear cryptanalysis over binary fields plays on the duality between $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$. The cipher can then be viewed as a vector Boolean function, and each ciphertext bit can be described as a Boolean function in (hopefully all) plaintext bits.

Since p is prime, field elements in $\mathbb{F}_p$ cannot be broken into smaller components in a dual to $\mathbb{F}_{2^n}$. This creates a problem for an adversary wishing to use linear cryptanalysis against *Rescue*. This is not to say that the cipher is necessarily resistant to all linear cryptanalysis-like attacks. A natural extension of linear cryptanalysis to $\mathbb{F}_p$ would be to search for a linear equation approximating the cipher as a linear m-variate polynomial over $\mathbb{F}_p$. However, we argue that: (i) this is not linear cryptanalysis, and more importantly (ii) such an attack should be considered in the category of algebraic attacks. In particular, our Gröbner basis analysis below does just that. It describes the cipher as a set of multivariate polynomials in $\mathbb{F}_p[x_1, \ldots, x_m]$ and tries to "linearize" them (in the same sense that this term is used in linear cryptanalysis over binary vector spaces). One difference between the two attacks is that a linear attack is statistical in nature: it approximates a nonlinear function with an error (*bias*; *correlation*) whereas the Gröbner bases attack represents it exactly at the expense of taking explicit account of the higher-order monomials. Still, the two attacks employ roughly the same mechanism and hence the Gröbner bases analysis presented below also applies here.

We stress that the argument provided here is not meant to be rigorous and that further research is required before the security against linear cryptanalysis is sufficiently understood for this use case. As *Rescue* lives in the intersection of two relatively underdeveloped research areas (namely, ciphers optimized for arithmetization and ciphers operating over prime fields) we advise users to consult their friendly neighbourhood cryptographer for a third-party evaluation of this cipher with respect to their particular use case.

Interpolation Attacks We look at polynomial descriptions of the cipher over $\mathbb{F}_p^m$ after several rounds. Due to the α-inverse power map being of high degree, two rounds of the cipher already attain the maximum polynomial degree p. Moreover, due to this power mapping, the polynomial expression is dense. From [20], we know that meet-in-the-middle variants of the interpolation attack are possible, however this attack becomes infeasible after three rounds.

Gröbner Bases Like for *Vision*, we provide equations encoding the preimage of the *Rescue* sponge function where a single unknown message block was absorbed and one known message block was squeezed out. In contrast to *Vision* it is now possible to fold equations across two steps in order to reduce the number of

variables and equations.[10]

$$\left\{ \begin{array}{ll} \left(M^{-1}(S_{2i-1} - K_{2i-1})\right)^{\alpha} - P - K_0 = 0 & i = 1 \\ M(S_{2i-1}^{\alpha}) + K_{2i-1} - \left(M^{-1}(S_{2i+1} - K_{2i+1})\right)^{\alpha} = 0 & i \in \{1, \ldots, N_b - 1\} \\ M(S_{2i-1}^{\alpha}) + K_{2N_b} - S_{2N_b} = 0 & i = N_b \end{array} \right\}$$

This encoding introduces m new variables and as many new equations of degree α per extra full round. The first half round introduces one variable and m equations, whereas the last half round introduces no variables and one equation. So along with m variables representing a single state to start from, we have in total $1 + mN_b$ variables and $1 + mN_b$ equations. If the system of equations were regular its Hilbert Series would be $HS(z) = \frac{\prod_{i=0}^{mN_b}(1-z^{\alpha})}{(1-z)^{mN_b+1}}$ and its degree of regularity $d_{reg} = (\alpha - 1)(mN_b + 1) + 1$. Experimentally, we find that for small m and N_b, the degree of the Gröbner bases output by a Gröbner basis attack (the *concrete degree of regularity*) satisfies $d_{con} = \lfloor d_{reg}/2 \rfloor$.[11] We extrapolate from this concrete-to-regular ratio of degrees of regularity to determine the necessary number of rounds. In particular, for a system of degree of regularity d_{con}, polynomials in $mN_b + 1$ variables have $\binom{mN_b+1+d_{con}}{mN_b+1}$ monomials, and squaring that number lower bounds the complexity of linear system solving for a square matrix with that dimension, and hence also bounds the complexity of a Gröbner basis attack.

4.4 Choosing the Number of Rounds

From the cryptanalysis of the previous sections we can deduce the number of rounds *Rescue* requires to resist known attacks. Again we see that the "classical" attacks become infeasible after a small number of rounds and that the bottleneck is the Gröbner basis attack.

From our analysis, we find that the base-2 logarithm of the attack complexity is lower-bounded by $4mN_b$. Accounting for a factor two security margin, we set the number of rounds to $2\lceil s/4m \rceil$ with a minimum of 10 rounds, for a security level of s.

Again, and similar to the case of *Vision*, we only experimented with Gröbner basis attacks for the case of hashing in sponge mode. In addition, the formula for the number of rounds was developed for $\alpha = 3$, which is the smallest α that can be used for *Rescue*. Using $\alpha > 3$ (when α is still a small prime) should increase the resistance against Gröbner basis attacks. Here too, we hope that third-party cryptanalysis can eventually inspire confidence in reducing the number of rounds.

5 AIR constraints for ZK-STARKs

In this section we analyze the efficiency of *Vision*, *Rescue*, and *MiMC* when described as a set of Algebraic Intermediate Representation (AIR) constraints.

[10] How folding is done for *Rescue* is further explained in Section 5.

[11] The experiments can be viewed in Appendix A.

A performance comparison is delayed until Section 8, where it is presented jointly with the analogous comparison but for Rank-One Constraint Satisfaction (R1CS) systems and masked MPC. For the sake of readers not versed in the relevant definitions related to STARKs [7] we recall those, along with a simple motivating example in Appendices B and C.

Notation. When describing AIRs we use the following conventions. Variables of the multivariate polynomials are denoted with capital letters $(X, K, R, \ldots)$. Plain variables denote the *current* state and primed variables (X', K', R') denote variables describing the state at the *next* cycle of the computation. We limit ourselves to constraints involving only two consecutive states. We use $[i, j]$ (or $[i]$) to select the indicated element from a matrix (resp. vector). When not affixed to a vector the notation $[m]$ is shorthand for the set $\{1, \ldots, m\}$. Furthermore, we extend set-builder notation to indicate multiple set members for each conditional satisfaction, *i.e.*, $\{a_i, b_i \mid i \in [2]\} = \{a_1, a_2, b_1, b_2\}$.

5.1 Encoding of a *Vision* Step as a Set of AIR Constraints

We present an AIR with $\mathtt{w} = 4m$, $\mathtt{t} = 2$ and degree $\mathtt{d} = 2$ for a single step of *Vision*. The sponge-based *Vision hash* replaces the key schedule with fixed constants, and hence has half the width of the cipher ($\mathtt{w} = 2m$) and the same length. We describe only the second step in the round in which $B(X)$ is used. The first step, which uses $B^{-1}(X)$, is analogous. First we deal with computing the key schedule, which requires $2m$ variables, denoted $K[1], \ldots, K[m]$ and $R[1], \ldots, R[m]$. Let $M[i, j]$ denote the (i, j)-entry of the MDS matrix M, let $C_k[i] \in \mathbb{F}_{2^{n/m}}$ be the ith field element of the kth step constant, and let $B(Z) = b_0 + b_1 Z + b_2 Z^2 + b_3 Z^4$ be the quartic polynomial used by *Vision*.

1. The first cycle is used to compute the map $x \mapsto x^{q-2}$, mapping x to its inverse when x is nonzero and otherwise keeping x unchanged. The following set of constraints (polynomials) ensures this,

$$\{K[i]K'[i] - R[i], K[i](1 - R[i]), K'[i](1 - R[i]) \mid i \in [m]\} \quad .$$

   To see this, notice that when $K[i] \neq 0$ the second constraint forces $R[i] = 1$ in which case $K'[i] = K[i]^{-1}$, and when $K[i] = 0$ the first constraint forces $R[i] = 0$ so the last constraint forces $K'[i] = 0$ as well.
2. The second cycle uses the auxiliary variable $R[i]$ to equal $K[i]^2$, and so, there exists a quadratic polynomial in $K[1], \ldots, K[m]$ and $R[1], \ldots, R[m]$ that computes the concatenation of the quartic polynomial B along with the linear transformation M and the addition of the step constant C_k used in the kth step. The following constraints ensure that $K'[1], \ldots, K'[m]$ hold the correct values, given $K[1], \ldots, K[m]$,

$$\left\{ R[i] - K[i]^2, K'[i] - \left(C_k[i] + \sum_{j=1}^{m} M[i, j] \left(b_0 + b_1 K[j] + b_2 R[j] + b_3 R[j]^2 \right) \right) \;\middle|\; i \in [m] \right\}.$$

A single step of the cipher is identical to the key schedule, with the main difference being that instead of adding a step constant (denoted C_k above) we add the kth key expansion during that stage. It follows that with an additional $2m$ variables and essentially the same set of constraints as above, we have accounted for the full AIR of the *Vision* round.

The *Vision hash* is a sponge construction and so the keys are fixed to certain known constants. The key schedule is dropped, leading to an AIR of width $\mathsf{w} = 2m$ and $\mathsf{t} = 2$ cycles per step.

Note that one *could* use different AIRs than described above to capture the same computation, just as we could use different AIRs to capture the Fibonacci computation of the example in Appendix B. For instance, one may increase the number of cycles per step from 2 to $2m$, while decreasing the width from $4m$ to 4 , by operating on the m state registers sequentially instead of in parallel. However, this alternative description does not reduce the overall size of the AET which stands at $8m$ per step (and $16m$ per round). Similar trade-offs can be applied to *Rescue*, as well, which we discuss next.

5.2 Encoding of a *Rescue* Step as a Set of AIR Constraints

Rescue is quite similar to *Vision* but simpler from an algebraic perspective. The main difference between the two ciphers is that the inverse step of *Vision* is replaced with a cubing operation (*i.e.,* $\alpha = 3$) and the quartic polynomial is removed. The result is that each step of the *Rescue* key schedule involves only m cubic polynomials (or inverses thereof), so we can encode it via an AIR using $\mathsf{d} = 3$ with a single cycle per step and width m. We denote the m variables of the $(2k+1)$th step by $K[1], \ldots, K[m]$, of the $2(k+1)$th step by $K'[1], \ldots, K'[m]$, and the ith element of the $(2k+1)$th step constant by $C_{2k+1}[i] \in \mathbb{F}_p$. The following constraints ensure that $K'[1], \ldots, K'[m]$ hold the correct values, given $K[1], \ldots, K[m]$,

$$\left\{ K'[i] - \sum_{j=1}^{m} C_{2k+1}[i] + \sum_{j=1}^{m} M[i,j]K[j]^3 \,\middle|\, i \in [m] \right\} .$$

The first step of each round has an analogous constraint set. We conclude that *Rescue* key schedule step AIR has degree $\mathsf{d} = 3$, state width $\mathsf{w} = m$ and $\mathsf{t} = 1$ cycles per step, and twice as many cycles per round.

The representation of the *Rescue* state function (not the key schedule) admits an optimization when one considers an adapted round as shown in Figure 7, in which the second step of one round is joined with the first step of the next, and in which the first and last step of the entire primitive are taken separately (and verified using the above encoding). We connect S and S' from the middles of rounds k and $k+1$ using m cubic equations effectively skipping the evaluation of the state after round k. The result is that we can encode the adapted round function via an AIR with a single cycle per round, $\mathsf{d} = 3$ and width m. The following constraints ensure that $S'[1], \ldots, S'[m]$ hold the correct values, given

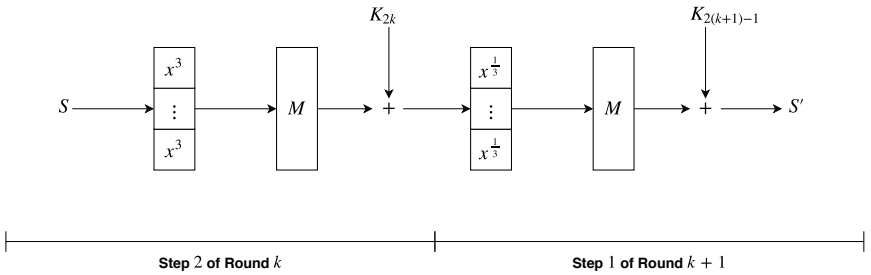


Figure 7: An adapted representation of a round of *Rescue* better suited for STARK evaluation.

$S[1], \ldots, S[m]$, $K_{2k}[1], \ldots, K_{2k}[m]$ and $K_{2(k+1)-1}[1], \ldots, K_{2(k+1)-1}[m]$,

$$\left\{ \sum_{j=1}^{m} M[i,j]S[j]^3 + K_{2k}[i] - \Big(\sum_{j=1}^{m} M^{-1}[i,j] \left(S'[j] - K_{2(k+1)-1}[j] \right) \Big)^3 \;\Big|\; i \in [m] \right\}.$$

We conclude that the *Rescue* state function AIR has degree $\mathsf{d} = 3$, state width $\mathsf{w} = m$ and $\mathsf{t} = 1$ cycle per round. When this permutation is used as a hash in sponge mode, *Rescue* does not require an AIR for the key schedule; this was also the case for *Vision*.

5.3 Encoding of a *MiMC* Round as a Set of AIR Constraints

The ℓth round of *MiMC*-q/q or *MiMC*-$2p/p$, with p and q a prime or a prime power of 2, maps a single variable x to $x^3 + C_\ell + K$ where C_ℓ is a round constant and the key K. Thus, we reach an AIR of degree $\mathsf{d} = 3$, width $\mathsf{w} = 1$ and $\mathsf{t} = 1$ cycle per round. Using X to denote the current state and X' the next state, the AIR (for the ℓth round) is the following simple construction:

$$\left\{ X' - (X^3 + C_\ell + K) \right\} \quad .$$

For *MiMC*-q/q this description can be used directly. For the case of *MiMC*-$2p/p$, variable names need to be remapped to account for addition between the two halves and the Feistel swap. Similar to *Rescue*, we can represent two rounds of *MiMC*-$2p/p$ (represented in Figure 8) using two cubic polynomials. The following constraints ensure that X'_L, X'_R hold the correct values, given X_L, X_R and K ,

$$\left\{ X_R + (X_L + K + C_\ell)^3 - X'_R,\ (X'_R + K + C_{\ell+1})^3 - X'_L \right\}.$$

As a result, we reach an AIR of degree $\mathsf{d} = 3$, width $\mathsf{w} = 2$ and $\mathsf{t} = 1$ cycle per two rounds.

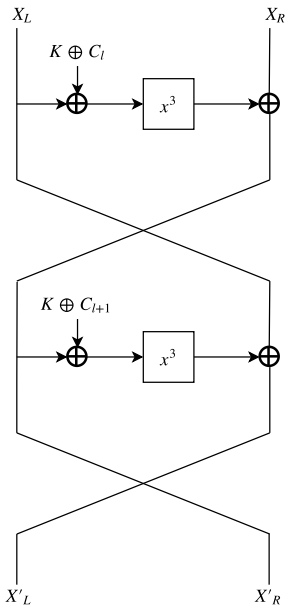


Figure 8: Two rounds of *MiMC*-$2p/p$.

6 Zero-Knowledge Proofs Based on R1CS Systems

In this section we evaluate the efficiency of *Vision*, *Rescue* and *MiMC* when encoded as rank one constraint satisfaction (R1CS) systems. Such systems are used by many zero-knowledge proof systems that operate on arithmetic circuits, such as Pinocchio [25], ZK-SNARK [10], Aurora [11], Ligero [3], and Bulletproofs [13].

6.1 Encoding of a *Vision* Step as a System of Rank-one Constraints

Recalling the two cycles of the AIR for *Vision* recounted earlier for constructing each of the key and round (section 5.1), we convert them into a system of R1CS constraints. Consider the key schedule first; the cipher round is identical. The first cycle is converted into $3m$ R1CS constraints. The second cycle splits the evaluation of the affine polynomial into two parts, each involving one squaring and thus m constraints for each part, resulting in a total of $2m$ constraints for the second cycle. For this latter constraint we notice that over binary fields (of size 2^k , integer k) it is the case that

$$\sum_j M[i,j]b_3 R[j]^2 = (\sum_j \alpha_j R[j])^2$$

for the constants α_j satisfying $\alpha_j^2 = M[i,j]b_3$. To see this, observe that in the expansion of the right hand side $\sum_j \sum_{j'} \alpha_j \alpha_{j'} R[j]R[j']$, the terms $\alpha_j \alpha_{j'} R[j]R[j']$

where $j \neq j'$ occur twice and thus disappear modulo 2. The constants α_j are guaranteed to exist because the map $x \mapsto x^2$ is bijective over binary fields. As a result, a squaring over a binary field requires only m constraints instead of m^2 constraints when working over non-binary fields. Since each step involves both the key derivation and the cipher step, we observe that the cost of a *Vision* block cipher step is $10m$ R1CS constraints, and that of a round is $20m$.

When used in sponge hash mode the key schedule is fixed, and so the number of R1CS constraints per step is halved. This gives a total number of $5m$ constraints per step (and twice that number per round.).

6.2 Encoding of a *Rescue* Step as a System of Rank-one Constraints

To efficiently encode a step of *Rescue* for $\alpha = 3$, we use two R1CS constraints to compute the cube of a state variable giving a total of $2m$ constraints for the cubing operations over the whole state. The step using the inverse cubing map is analogous. The linear combinations due to the MDS matrix M can be integrated into these $2m$ constraints. Since the same computation is applied to the key schedule when used as a cipher, we count $4m$ per step, twice as many constraints ($8m$) per round, and $2m$ constraints per step for *Rescue* used in sponge hash mode because the key schedule is fixed.

6.3 Encoding of a *MiMC* Round as a System of Rank-one Constraints

Since a single *MiMC* round involves a cubic polynomial, it can be seen that 2 R1CS constraints suffice.

7 MPC with Masked Operations

In this section we explore how to implement *Vision* and *Rescue* over MPC using masked operations.[12] We consider three masked operation techniques: one technique to find the inverse of a shared field element due to Bar-Ilan and Beaver [4]; one technique to raise a shared element to an arbitrary but known power due to Damgård *et al.* [17]; and one new technique to compute the compositional inverse of a low-degree linearized polynomial.

The common strategy behind these techniques is to apply random and unknown masks to a shared secret value and opening their sum. The operation proper is applied to the opened variable giving a known but still-masked output value. The mask on this output value is then removed by combining it with the output of a dual operation applied to the original shared random mask. The benefit of these techniques comes from shifting the computation of this mask and its dual to the offline phase, which is possible as this computation does not

[12] For the sake of completeness, we also consider MPC implementations based on the more straightforward square-and-multiply algorithm in Appendix E.

depend on the value to which the operation is applied. In the online phase, the regular operation is computed locally (*i.e.* without needing to communicate); the dual operation does require communication but it is cheaper.

The first two of these techniques require zero-tests — sub-protocols that produce a sharing of 1 if its input is a sharing of 0, and a sharing of 0 otherwise. For binary fields and XOR-based secret sharing, the bit decomposition of a shared element imposes no additional cost. Moreover, using a tree-based multiplication strategy, the output of the zero-test can be computed in a logarithmic number of communication rounds. For large prime fields, or binary fields and generic secret sharing, an observation by Nishide and Ohta provides the protocol with a similar list of bits with only two rounds of interaction [23]; the same tree-based strategy then computes the zero-test output once again with a logarithmic number of communication rounds. Our MPC implementations of *Rescue* and *Vision* are agnostic of the particular zero-test as well as of the secret sharing mechanism, and the selection of the constituent sub-protocols depends exclusively on the trade-offs imposed by the application at hand.

It is possible to drop the zero-test altogether; this produces a method which evaluates an inversion in a small constant number of communication rounds. The method then *assumes* that the value is non-zero. In a passive adversarial model, the probability of this event may be cryptographically small enough to neglect, especially if the field size is chosen in excess of the bare security requirement. Conversely, an active adversary may try to manipulate the event and force a zero value into the S-box. We therefore recommend that users drop the zero-test only after thoroughly investigating its implications.

7.1 *Vision*

Recall that elements of the state in *Vision* are members of the extension field $\mathbb{F}_{2^{n/m}}$. Since we use a linear secret sharing scheme, we can perform the additions and multiplications-by-constants from *Vision* in a straightforward manner, namely by manipulating shares locally. In particular, this means that applications of the MDS matrix to the working state impose no extra cost. However, nonlinear operations do not admit such a straightforward realization and instead require creative solutions to retain an efficient implementation.

Inversion Only two component blocks of *Vision* induce a cost: the inversion operation, and the polynomial evaluation of B and B^{-1}. All other operations are linear and thus free. Recall that the state of *Vision* consists of m field elements. Therefore, each round includes m initial inversions, m inverse-polynomial evaluations, followed by another m inversions and m regular polynomial evaluations. These m executions are independent and can therefore be performed in parallel. The cipher consists of N_b rounds in total. The key schedule algorithm doubles these numbers, but its cost can be amortized over the entire execution of the protocol so we neglect it here.

To evaluate the inversion step, we use the technique due to Bar-Ilan and Beaver [4], of which pseudocode is given below. In a nutshell, the protocol starts

with some secret shared random value $[r]$, generated *e.g.*, using PRSS [14]. Generating this randomness does not require any communication between the parties. The parties then multiply this random value with the proper input $[x]$, thereby obtaining the shared value $[rx]$, which is then opened to reveal rx. The inversion is computed locally by all parties: $(rx)^{-1}$. A subsequent multiplication of this known constant with the secret random value $[r]$ removes the mask. This produces $(rx)^{-1}[r] = [x^{-1}r^{-1}r] = [x^{-1}]$, which is exactly the sharing of the inverse of x. This technique only works of both x and r are nonzero, which explains where the required zero-tests come from.

The procedure above requires 2 communication rounds and works for all non-zero elements $x \in \mathbb{F}_{2^n}$. In scenarios where the shared value is unlikely to be zero (*i.e.*, if the field is large enough), this technique can be used directly. Ignoring the zero test, the total cost of this method is 1 communication round: it is possible to merge a multiplication and an opening call.

For the zero-test we adopt an observation by Nishide and Ohta [23]: we produce n/m secret-shared random bits $[r_i]$ jointly comprising the binary representation of a random mask $[r]$ with which we mask $[x]$. Then we open $[c] = [x] + [r]$ and obtain this number's binary representation. Note that $c = r$ if and only if $x = 0$. It follows that by flipping the bits $[r_i]$ whose counterparts in c are 0, we obtain a string of shared 1's iff $c = r$. Multiplying these bits together in the tree-based multiplication strategy gives the zero-test output. The generation of random bits requires 1 offline communication round. The tree-based multiplication strategy involves $n/m - 1$ multiplications in $\lceil \log_2(n/m) \rceil$ online rounds. The total cost of this zero-test is therefore

$$\begin{aligned} &\#\text{ offline rounds: } \quad 1 \ , \\ &\#\text{ online rounds: } \quad \lceil \log_2(n/m) \rceil \ , \\ &\#\text{ multiplications: } m \cdot 2 \cdot (n/m) - 1 \ . \end{aligned}$$

Inverse of Sparse Linearized Polynomial A similar approach can be used to compute $B^{-1}(x)$ thanks to the following observations.[13] We ignore for the sake of simplicity the constant that makes $B(x)$ affine and not linear (over $\mathbb{F}_2$); this simplification makes $B^{-1}(x)$ linear also. In particular, this means that $B^{-1}(x + y) = B^{-1}(x) + B^{-1}(y)$. Noting that $B(x)$ consists of three terms with degrees 1, 2, and 4, we can calculate the output $[B^{-1}(x)]$ from $[x]$ as follows: create a shared random mask $[r]$ and compute $[B(r)]$. Then open $[x - B(r)]$ and apply B^{-1} locally to this opened value. Then adding $[r]$ back gives $B^{-1}(x - B(r)) + [r] = [B^{-1}(x) - r + r] = [B^{-1}(x)]$, which is exactly the desired output. Note that the evaluation of $B(r)$ is not tied to any input data, and can therefore be pre-computed in an offline phase. The pseudocode in Appendix D.2 shows this procedure more formally.

Total cost The implementation of a round of *Vision* follows straightforwardly from using these building blocks, along with linear (and thus local) operations.

[13] To the best of our knowledge, this technique is new and thus an independent contribution of this paper.

A round of *Vision* consists of 2 calls to the inversion protocol at a total cost of 2 communication rounds (ignoring the zero-test), the evaluation of $B^{-1}(x)$ with an overall cost of 3 communication rounds (2 of which are be precomputed in an offline phase), and the evaluation of $B(x)$ at a cost of 2 communication rounds. While these elements are performed on each of m elements, they are performed independently and are hence parallelizable. The total complexity of *Vision* is therefore

$$\begin{aligned} &\#\text{ offline rounds: } \quad 2 \ , \\ &\#\text{ online rounds: } \quad 2+1+2=5 \ , \\ &\#\text{ multiplications: } m\cdot(2+3+2)=m\cdot 7 \ . \end{aligned}$$

7.2 *Rescue*

The only nonlinear operations of *Rescue* to take into account are the α and inverse-α power maps. To achieve this, We have adapted, for any arbitrary large α, the exponentiation technique introduced by Damgård *et al.* [17]. This way, we can offload a portion of the computation to an offline phase and retain a constant online complexity (i.e., 1 round). A small adaptation of this technique computes the inverse power map at the same online cost. We summarize this adaptation here.

The participants generate a shared secret mask $[r]$. They compute $[r^\alpha]$ and $[r^{-1}]$ in the offline phase. In the online phase, they open the masked value $[xr^\alpha]$ and locally raise this known value to the power $1/\alpha$. At this point, a simple multiplication-by-constant yields $(xr^\alpha)^{1/\alpha}[r^{-1}] = [x^\alpha r r^{-1}] = [x^\alpha]$. The pseudo-code for both procedures is shown in Appendix D.3.

Each procedure requires $\lceil \log_2 \alpha \rceil + 2$ multiplications in total, and $\lceil \log_2 \alpha \rceil + 2$ communication rounds (including the 1 online round). In the case of the inverse alpha map, obtaining $[r^{-1}]$ can be combined with the exponentiation, thus reducing by one the number of communication rounds. All operations on r can be executed in parallel during an offline phase as they do not depend on the input and on each other.

The implementation of *Rescue* is now straightforward. Each power map is applied in parallel to all m elements of the state. The multiplication with the public MDS matrix is free. The cost of a single round is therefore

$$\begin{aligned} &\#\text{ offline rounds: } \quad \lceil \log_2 \alpha \rceil + 1 \ , \\ &\#\text{ online rounds: } \quad 2 \ , \\ &\#\text{ multiplications: } 2m\cdot(\lceil \log_2 \alpha \rceil + 2) \ . \end{aligned}$$

Note that we have excluded the zero-test from this analysis, given the relatively large field. However, when the risk for having $x = 0$ is non-negligible, the additional cost of testing for zero should be taken into account. The protocol by Nishide and Ohta [23] performs this test by subtracting from $[x]$ a random shared value $[r]$ with known bit decomposition $[r_0], \ldots, [r_{\lceil \log_2 p \rceil - 1}]$. The protocol opens $[x-r]$ and locally computes this number's binary expansion, and each such bit is subtracted from its corresponding $[r_i]$. The protocol is now in possession

of $\lceil \log_2 p \rceil$ shared secrets which are all zero iff $x = 0$. A tree-based multiplication strategy completes the zero-test. The generation of a bit-wise uniform $[r]$ in p can be done in 7 communication rounds and $56 \cdot \log_2(p)$ multiplications if $\log_2(p) \geq 36$ (see [22]). These operations add an additional cost of

$$\begin{aligned} &\#\text{ offline rounds: } \quad 7 \ , \\ &\#\text{ online rounds: } \quad \lceil \log_2(\lceil \log_2(p) \rceil) \rceil \ , \\ &\#\text{ multiplications: } m \cdot (56 \cdot \lceil \log_2(p) \rceil + \lceil \log_2(p) \rceil - 1) \ . \end{aligned}$$

7.3 *MiMC*

MiMC works by repeatedly cubing an element and interleaving this cube map by constant and key injections. For decryption, the inverse of this cube map is required. However, we have already discussed both nonlinear operations in our description of *Rescue* as they constitute the special case where $\alpha = 3$. Therefore, a single round of *MiMC* has a complexity of

Encryption:		Decryption:	
# offline rounds:	$\lceil \log_2 \alpha \rceil + 1 \approx 3$	# offline rounds:	$\lceil \log_2 \alpha \rceil \approx 2$,
# online rounds:	1	# online rounds:	1 ,
# multiplications:	$\lceil \log_2 \alpha \rceil + 2 \approx 4$	# multiplications:	$\lceil \log_2 \alpha \rceil + 2 \approx 4$.

Note that now there is once again the possible security hazard when the secret value, that is undergoing the inverse cube map, is zero. The protocol should either perform the zero-test and incur its associated cost; or ignore it and argue why the security degradation is negligible or tolerable.

8 Comparison

To compare *MiMC* with *Vision* and *Rescue*, we set $m = 2$, $n = 128$, $p = 2^{64}+13$, $q = 2^{127} + 45$ and $\alpha = 3$. For the purpose of the present comparison, the number of AIR constraints (of degree d) is given by the value of $\mathsf{w} \cdot \mathsf{t}$, we ignore the zero-test for MPC and observe that the offline parts can be done in parallel for all rounds.

We compare the three algorithms for AIR (Table 1), R1CS (Table 2) and masked MPC (Table 3) in two scenarios: as block ciphers and as sponge functions. For 128-bit block cipher security, we require 24 rounds of *Vision*; 32 rounds of *Rescue*; 82 rounds of *MiMC*-q/q; and 164 rounds of *MiMC*-$2p/p$. Recall that *MiMC*-$2p/p$ is better suited for arithmetization in hash mode (see Section 2.4). In sponge mode these parameters offer only 32 bits security against collisions; nevertheless they allow for an apples-to-apples comparison with the same rate and capacity. Note that the field size does not change the cost under the metrics we consider in this paper.

Table 1: Comparison of *Vision*, *Rescue*, *MiMC-q/q* and *MiMC-2p/p* over AIR. With $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{127} + 45$.

	Mode	Degree (d)	Width (w)	Cycles (t)	$\mathsf{w} \cdot \mathsf{t}$
Vision	BC	2	8	96	768
Rescue	BC	3	2	97	194
MiMC-q/q	BC	3	1	82	**82**
Vision	Hash	2	4	96	384
Rescue	Hash	3	2	33	**66**
MiMC-2p/p	Hash	3	2	82	164

Table 2: Comparison of *Vision*, *Rescue*, *MiMC-q/q* and *MiMC-2p/p* over R1CS. With $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{127} + 45$.

	Mode	Constraints per Step	Steps	Total
Vision	BC	20	48	960
Rescue	BC	8	64	512
MiMC-q/q	BC	2	82	**164**
Vision	Hash	10	48	480
Rescue	Hash	4	64	**256**
MiMC-2p/p	Hash	2	164	328

Table 3: Comparison of the *Vision*, *Rescue*, *MiMC-q/q* and *MiMC-2p/p* over MPC using masked operations. With $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{127} + 45$.

	Mode	Offline Rounds	Online Rounds	Multiplications
Vision	BC	2	120	336
Rescue	BC	3	**64**	512
MiMC-q/q	BC	3	82	**328**
Vision	Hash	2	120	**336**
Rescue	Hash	3	**64**	512
MiMC-2p/p	Hash	3	164	656

9 Conclusion

This paper introduced *Marvellous* —a universe of cryptographic primitives optimized for arithmetic computation. The universe consists of two families of block ciphers: *Vision* operates on elements in $\mathbb{F}_{2^{n/m}}^m$ and *Rescue* operates on elements in $\mathbb{F}_p^m$. Both can be turned into a hash function by using the sponge construction and by fixing the key input to zero. We compare our ciphers to *MiMC* in terms of efficiency with respect to three use cases: scalable and transparent zero-knowledge argument of knowledge (STARK) systems, zero-knowledge proof systems based on rank-one constraint satisfaction (R1CS) systems, and multiparty computation (MPC) with masked operations.

The *Marvellous* designs operate on vectors of $m > 1$ elements, which puts them at a disadvantage compared to primitives operating on a single field element ($m = 1$) such as *MiMC* or *Jarvis* in terms of efficiency. We did consider the counterparts of *Vision* and *Rescue* with $m = 1$. However, experimental results with $m = 1$ indicate that the degree of regularity grows poorly in the number of rounds. The purpose of the parameter constraint $m \geq 2$ is to counteract this vulnerability. We plan to investigate this surprising observation further and experimentally assess the security of designs having $m = 1$.

In addition, the *Marvellous* designs may offer advantages beyond merely the availability of alternatives with the same functionality to fall back on in the event of a break. The parameter m can be optimized according to the application's native support for parallelism. Furthermore, having fewer rounds as a result of increasing m is conducive to better latency. Lastly, *Marvellous* offers the designer some flexibility to adapt the primitive's field to the context.

Finally, we stress the highly conservative approach for determining the number of rounds. We expect that third party cryptanalysis concludes, after due deliberation, that the number of rounds can be safely reduced in order to improve the efficiency even further. Aggressive optimization is likely to further reduce the cost of actual implementations in real-world scenarios.

Acknowledgments The authors would like to thank Vincent Rijmen and Daira Hopwood for their useful comments.

This research was partly funded by Starkware Industries Ltd., as part of an Ethereum Foundation grant activity. The first author was also supported by Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070. The second author was supported by the Research Council KU Leuven, C16/18/004. Author 4 is supported by a Ph.D. Fellowship from the Research Foundation - Flanders (FWO). Author 5 was supported by an IWT doctoral grant and by the Nervos Foundation. These supports are greatly appreciated.

References

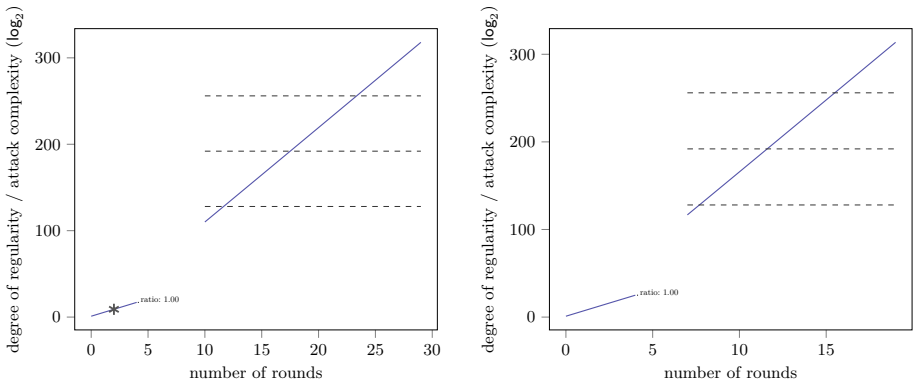
1. Albrecht, M., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. Cryptology ePrint Archive, Report 2016/687 (2016), `http:`

//eprint.iacr.org/2016/687
2. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: ASIACRYPT 2016, Part I. pp. 191–219. LNCS (2016). https://doi.org/10.1007/978-3-662-53887-6_7
3. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: ACM - CCS 2017 (October 2017)
4. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: ACM Symposium on Principles of Distributed Computing 1989. pp. 201–209 (1989). https://doi.org/10.1145/72981.72995
5. Barreto, P.S.L.M., Rijmen, V.: Whirlpool. In: Encyclopedia of Cryptography and Security, 2nd Ed. pp. 1384–1385 (2011). https://doi.org/10.1007/978-1-4419-5906-5_626, https://doi.org/10.1007/978-1-4419-5906-5_626
6. Ben-Sasson, E., Bentov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., Virza, M.: Computational integrity with a public random string from quasi-linear pcps. In: EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 551–579 (2017). https://doi.org/10.1007/978-3-319-56617-7_19
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046
8. Ben-Sasson, E., Chiesa, A., Forbes, M.A., Gabizon, A., Riabzev, M., Spooner, N.: On probabilistic checking in perfect zero knowledge. arXiv preprint arXiv:1610.03798 (2016)
9. Ben-Sasson, E., Chiesa, A., Gabizon, A., Virza, M.: Quasilinear-size zero knowledge from linear-algebraic PCPs. In: TCC 2016. pp. 33–64. LNCS (2016)
10. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: CRYPTO 2013. pp. 90–108. LNCS (2013)
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. Cryptology ePrint Archive, Report 2018/828 (2018), http://eprint.iacr.org/2018/828
12. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the indifferentiability of the sponge construction. In: EUROCRYPT 2008. pp. 181–197 (2008). https://doi.org/10.1007/978-3-540-78967-3_11
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Efficient range proofs for confidential transactions. Cryptology ePrint Archive, Report 2017/1066 (2007), http://eprint.iacr.org/2017/1066
14. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Theory of Cryptography, pp. 342–362. LNCS, Springer (2005). https://doi.org/10.1007/978-3-540-30576-7_19
15. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: FSE 1997. pp. 149–165. LNCS (1997). https://doi.org/10.1007/BFb0052343
16. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002). https://doi.org/10.1007/978-3-662-04722-4
17. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: TCC 2006. pp. 285–304 (2006)
18. Damgård, I., Keller, M.: Secure multiparty AES. In: FC 2010, Tenerife. pp. 367–374. LNCS (2010). https://doi.org/10.1007/978-3-642-14577-3_31

19. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645 (2013). https://doi.org/10.1007/978-3-642-38348-9_37
20. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: FSE 1997. pp. 28–40. LNCS (1997). https://doi.org/10.1007/BFb0052332
21. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. In: Foundations of Computer Science 1990. pp. 2–10. IEEE (1990)
22. Ning, C., Xu, Q.: Constant-rounds, linear multi-party computation for exponentiation and modulo reduction with perfect security. In: ASIACRYPT 2011. pp. 572–589. LNCS (2011). https://doi.org/10.1007/978-3-642-25385-0_31
23. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: PKC 2007. pp. 343–360 (2007)
24. Nyberg, K.: Differentially uniform mappings for cryptography. In: EUROCRYPT 1993. pp. 55–64. LNCS (1993). https://doi.org/10.1007/3-540-48285-7_6
25. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy 2013. pp. 238–252. Oakland '13 (2013)
26. Razborov, A.A.: Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. In: Mathematical notes of the Academy of Sciences of the USSR. vol. 41 - 4, pp. 333–338 (1987)
27. Szepieniec, A., Dhooghe, S.: Marvellous (instance generator) (2019), `https://github.com/KULeuven-COSIC/Marvellous.git`
28. Wahby, R.S., Setty, S., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: NDSS 2015. LNCS (2015)

A Experimental Results Using Gröbner Bases

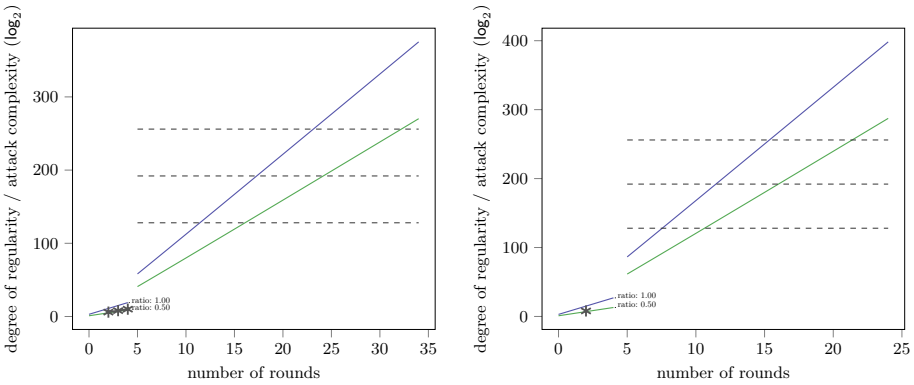
Vision. Due to the high complexity of calculating the degree of regularity (*i.e.*, of performing the Gröbner basis calculation and observing the degree of the resulting basis) even for round reduced versions, we have few results even after running the experiment for 60 hours. The one observed data point, coupled with the prohibitive complexity of obtaining more, justify the assumption that the attacked system behaves like a regular system of the same number of equations and variables. We extrapolate this finding and show the complexity of constructing a degree reverse lexicographic Gröbner basis of *Vision* providing for a different number of rounds and parameters m. We found these results were independent of the field size.



(a) $m = 2$ (b) $m = 3$

Figure 9: Experimental results of round reduced *Vision* for parameters m. The bottom left graphs show on the vertical axis the degree of regularity with experiments denoted by asterisks. The upper graph shows the resulting complexity of constructing a Gröbner basis assuming the system is regular with the grey dotted lines showing $128, 192$ and 256-bits of complexity.

Rescue. We made the same experiments for *Rescue*. We calculated the degree of the Gröbner basis output by the Gröbner basis algorithm for several round-reduced versions of *Rescue* and found that this concrete degree was exactly half the degree of regularity of regular systems, independently of the field size. We show the complexity of constructing a degree reverse lexicographic Gröbner basis of round reduced versions of *Rescue* for different m assuming the same concrete-to-regular degree ratio holds even for larger round numbers. For comparison, we also show the complexity if the system were regular.



(a) $m = 2$ (b) $m = 3$

Figure 10: Experimental results of round reduced *Rescue* for parameters m. The bottom left graphs show on the vertical axis the degree of regularity of regular systems (in blue), and half that number (in green), with experimental observations denoted by asterisks. The upper graph shows the resulting complexity of constructing a Gröbner basis with the grey dotted lines showing $128, 192$ and 256-bits of complexity.

B STARK Intuition

We start be recalling the relevant definitions from [7], along with a simple motivating example.

Scalable Interactive Oracle Proofs (IOPs) and Transparent Arguments of Knowledge (STARKs) like [6, 7] express computations using an *Algebraic* Execution Trace (AET): for a computation with t steps and internal state captured by w registers, the trace is a $\mathsf{t} \times \mathsf{w}$ array. Each entry of this array is an element of a finite field $\mathbb{F}$.

Before presenting formal definitions, we motivate them using a simple example. Suppose the prover wishes to prove the statement below, where p is prime and $\mathbb{F}_p$ is the finite field of size p:

> "$\exists x_0, x_1 \in \mathbb{F}_p$ such that y is the qth element in the Fibonnacci sequence defined recursively for $i > 1$ by $x_i = x_{i-1} + x_{i-2} \mod p$."

An execution trace *proving* the statement above is a $(q+1) \times 1$ array in which the ith state is, supposedly, x_i. Now, to verify the correctness of the statement our verifier must check that the following two conditions hold:

- **boundary constraints:** the last entry equals y.
- **transition relation constraints:** for each $i \leq q-1$, the i^{th} register plus the $i{+}1^{\text{st}}$ register equals the $i{+}2^{\text{nd}}$ register. This can be captured succinctly by a constraint of the form

$$X_{\mathsf{current}} + X_{\mathsf{next}} - X_{\mathsf{next\_next}} = 0 \ ,$$

which is applied to each consecutive triple-of-states in the trace. Satisfying a constraint always means setting it to 0, so the right hand side above is redundant and henceforth we shall simplify such a constraint and write only its left hand side, namely,

$$X_{\mathsf{current}} + X_{\mathsf{next}} - X_{\mathsf{next\_next}} \ .$$

Alternatively, the execution trace could be a $q \times 2$ array in which the ith state supposedly contains x_i, x_{i+1}. Now, the verifier checks two constraints for each pair of consecutive states, described next by using X, Y to denote the two registers capturing the state,

$$(i)\ X_{\mathsf{current}} + Y_{\mathsf{current}} - Y_{\mathsf{next}}; \quad (ii)\ X_{\mathsf{next}} - Y_{\mathsf{current}} \ .$$

The boundary constraint would now check that the $[q, 2]$-entry of the execution trace equals y.

Comparing the two solutions above, we see that the second one is $\times 2$ bigger than the first, but its constraints involve only two consecutive states, rather than three states required in the first solution. The second solution also has a larger set of constraints (two constraints vs. one constraint in the first solution) but in both solutions all constraints are multivariate polynomials of degree 1. The main takeaway message here is that the same computation can be expressed in several ways via different execution traces and constraint systems.

C Formal Description of an Algebraic Execution Trace

We start with the definition of an algebraic execution trace.

Definition 1 (Algebraic Execution Trace (AET)). *An* Algebraic Execution Trace *(AET) of* width w *and* length t *over a field* $\mathbb{F}$ *is an array with* t *rows and* w *columns, each entry of which is an element of* $\mathbb{F}$*. The* i*th row represents the* state *of a computation at time* i *and the* j*th column represents an* algebraic register*. The* size *of the AET is* $\mathsf{t} \cdot \mathsf{w}$*.*

Next, we define a constraint system that checks whether an execution trace is valid with respect to a computation. Informally, the constraints capture the transition relation of the computation, each constraint is a polynomial, and an assignment satisfies a constraint iff the constraint (polynomial) evaluates to 0 under the assignment.

Definition 2 (Algebraic Intermediate Representation (AIR)). *An Algebraic Intermediate Representation (AIR) of degree* d*, width* w *and length* t *over the field* $\mathbb{F}$ *is a set of multivariate polynomials of total degree at most* d*, with coefficients in* $\mathbb{F}$ *and variable set* $R_{ij}, i \leq \mathsf{w}, j \leq \mathsf{t}$*.*

We point out that the definition of AIR in [7] is slightly more complicated (dealing with boundary constraints and neighborhood sets) but for the purpose of the current work the simpler definition above suffices.

D Algorithms for Masked MPC implementations

We provide here C++-like algorithms for the various masking techniques used in Section 7.

D.1 Inversion

```
Invert(x,n) {
       b = (x == 0); // log2(x) com calls
       c = 0;
       while(c == 0) {
              r = share_random();
              temp = (b + x);
              temp = temp * r;  // 1 com call
              c = open(temp);  // 1 com call
       }
       c = pow(c,2^n-2);
       c = (r * c) - b;

       return c;
}
```

D.2 Inverse of Sparse Linearized Polynomial

```
Invert_B(x) {
       r = share_random(); // offline
       b_r = B(r); // trivial impl. of B (2 rounds) -- and offline
       c = x + b_r;
       c = open(x + b_r); // 1 round
       c = B_inv(c); // B^-1(x + B(r))
       c = c - r; // B^-1(x) + B^-1(B(r)) - r
       return c;  // B^-1(x)
}
```

D.3 α-power and Inverse-α-power

```
AlphaPower(x,alpha) {
    c = 0;
    while(c == 0) {
    // offline phase
        r = share_random();
        rinv = Invert(r);
        rexp = rinv^alpha;
    // online phase
        c = open(x * r);
    }
    c = pow(c,alpha);
    c = c * rexp;
    return c;
}
```

```
InverseAlpha(x,alpha,alpha_inv) {
    c = 0;
    while(c == 0) {
    // offline phase
        r = share_random();
        rinv = Invert(r); //1 round
        rexp = r^alpha; //lg(alpha)
    // online phase
        c = open(x * rexp);
    }
    c = pow(c,alpha_inv);
    c = c * rinv;
    return c;
}
```

E MPC with Square-and-Multiply

In Section 7 we presented a method to compute families of the *Marvellous* universe in an MPC protocol using masked operations. In this section, we explore a more straightforward alternative: square-and-multiply. The main difference with our masked approaches relies on the fact that there is no mechanism to outsource computation to a pre-processing phase. In fact, all multiplications have to be performed online.

The exponents for all nonlinear operations for both *Vision* and *Rescue* (as well as *MiMC*) are publicly available. The complexity of square-and-multiply over MPC is upper bounded by 2ℓ multiplications and ℓ communication rounds, where ℓ is the bit length of the exponent. We refer the reader to Damgård and Keller for a more detailed treatment on the use of square-and-multiply over MPC [18]. For the treatment here, we assume the existence of a functionality `square_multiply(x,e)` that takes a shared secret $[x]$ and outputs $[x^e]$ with the stated complexity.

Vision. To implement the inversion of *Vision* using square-and-multiply, observe that $x^{-1} = x^{2^{n/m}-2}$ and no zero-test is required; the inverse can therefore be computed with `square_multiply(`$[x]$`,` $2^{n/m}-2$`)`. The complexity of this exponentiation is thus n/m communication rounds and $2n/m$ multiplications. The evaluation of B and of B^{-1} requires 2 and $n/m-1$ sequential multiplications, respectively. The linear components of *Vision* do not contribute to its cost. The total complexity of one round of this implementation of *Vision* is therefore

$$\begin{aligned} &\#\text{ offline rounds: } \ 0 \ , \\ &\#\text{ online rounds: } \ 3n/m+1 \ , \\ &\#\text{ multiplications: } m\cdot(5n/m+1) \ . \end{aligned}$$

Rescue. For *Rescue*, the use of square-and-multiply does not require any specific protocol adaptation. Both power maps can be obtained from an invocation of `square_multiply(`$[x]$`,`e`)`, where $e = \alpha$ or $e = \alpha^{-1} \mod p-1$. Like for *Vision*, the linear components do not contribute to the cost. Consequently, the total complexity of one round of this implementation of *Rescue* is

$$\begin{aligned}
&\#\text{ offline rounds: } 0 \ , \\
&\#\text{ online rounds: } \lceil \log_2 \alpha \rceil + \lceil \log_2 p \rceil \ , \\
&\#\text{ multiplications: } 2m \cdot (\lceil \log_2 \alpha \rceil + \lceil \log_2 p \rceil) \ .
\end{aligned}$$

MiMC. The most straightforward implementation of *MiMC* encryption is already using square-and-multiply. For the decryption mode, inverse cube map can be computed with square-and-multiply by invoking `square_multiply(`$[x]$`,`e`)`, where $e = 3^{-1} \mod q-1$. The total cost of one round of *MiMC* in this implementation is therefore

Encryption:	Decryption:
# offline rounds: 0	# offline rounds: 0 ,
# online rounds: 2	# online rounds: $\lceil \log_2 q \rceil$,
# multiplications: 2	# multiplications: $2\lceil \log_2 q \rceil$.

Comparison. Like before, we consider 24 rounds of *Vision* with $n = 128$ and $m = 2$; 34 rounds of *Rescue* with $p = 2^{64} + 13$, $\alpha = 3$ and $m = 2$; and 82 rounds of *MiMC*; each a parameter set targeting 128 bits of security. This consideration gives rise to the following table of comparison.

Table 4: Comparison of *Vision*, *Rescue*, and *MiMC* over MPC using square-and-multiply.

	Vision	*Rescue*	*MiMC* Enc.	*MiMC* Dec.
offline rounds	0	0	0	0
online rounds	4632	4420	164	10496
multiplications	15408	17680	164	20992