Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols

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 traditional symmetric primitives like 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*.
In this paper we study the design of secure cryptographic primitives optimized to minimize this metric. We begin by identifying the differences in the design space between such *arithmetization-oriented ciphers* and traditional ones, with particular emphasis on the available tools, efficiency metrics, and relevant cryptanalysis. This discussion highlights a crucial point — the considerations for designing arithmetization-oriented ciphers are fundamentally different from the considerations arising from traditional cipher design.
The natural next step is to identify sound principles to securely navigate this new terrain, and to materialize these principles into concrete designs. To this end, we present two families of arithmetization-oriented symmetric-key primitives. By motivating our design decisions at length with respect to the identified principles, we show that it is possible to design secure and efficient primitives for this emerging domain.
These primitives — *Vision* and *Rescue* — are benchmarked 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). These benchmarks show that our ciphers achieve a highly compact algebraic description, and thus benefit the advanced cryptographic protocols that employ them.

Keywords: Vision · Rescue · Marvellous · arithmetization · zero-knowledge proof · STARK · R1CS · MPC · Gröbner basis · sponge

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 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 AES, 3DES, SHA2-256, SHA3/Keccak, is well understood and widely agreed upon, 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], *MiMC* [3], and *Jarvis* [5]. The distinguishing feature of these ciphers is the alternative target for optimization: running time, gate count, memory footprint, power consumption, are all left by the wayside in favor of the number of non-trivial arithmetic operations. These ciphers can thus be characterized as *arithmetization-oriented*, as opposed to *traditional* ciphers which do not have the alternative optimization target.

Arithmetization-oriented cipher design should not be understood from the perspective of traditional cipher design. The relevant attacks and security analyses are different. Traditional constructions and modes of operation must be lifted to the arithmetic setting and their security proofs must be redone. The target applications are different and provide the designer with a new collection of tools to secure their design against attacks without adversely affecting efficiency. This efficiency is captured in terms of arithmetic metrics that vary subtly by applications — but jointly stand in stark contrast to traditional efficiency metrics such as the ones mentioned above.

As a field in its own right, the design of arithmetization-oriented ciphers is in its nascency. Rather than blindly optimize for a single vaguely defined metric and shipping the resulting construction as soon as possible, it is worthwhile and timely to stop and re-evaluate formerly optimal strategies with respect to this new field. The contribution of this work is not just the proposal of two new ciphers, although that was — and still is — certainly its motivation. The more important contribution consists of the steps taken towards a more systematic exploration and mapping of the problem and design landscape that these ciphers inhabit. Our ciphers, *Vision*[4] and *Rescue*[5] , merely represent the *Marvellous* culmination of our journey.

This paper is structured in accordance with a progressive refinement of focus. First, in Section 2, we characterize the common features of the advanced cryp-

[4] In the Marvel comics, Vision is a binary field powered android created by Tony Stark.

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

tographic protocols that arithmetization-oriented ciphers cater to and identify and clarify the exact and various efficiency metrics that are relevant in those contexts. Next, in Section 3 we explore the space of design considerations. In particular, we identify important differences (compared to standard symmetric cipher design) in terms of the security analysis as well as in terms of the tricks and techniques that can be employed in order to marry security with efficiency. Having surveyed the design space we then motivate our position in Section 4; here we provide concrete answers to questions raised in the preceding sections regarding the security rationale, potential pitfalls, and application constraints. Lastly, in Sections 5 and 6 we present the logical consequence of these design decisions: concrete specifications for *Vision* and *Rescue*, two families of the *Marvellous* universe.

We use three advanced cryptographic protocols as running examples of applications, and guiding beacons, throughout this paper: zero-knowledge proof systems for the Turing or RAM models of computation, for the circuit model of computation, and multi-party protocols. In particular, our discussion characterizes all three as arithmetic modalities of computation. We spend ample time identifying the correct efficiency metrics along with non-trivial design tools that these protocols enable. Finally, in Section 7 we compare *Vision* and *Rescue* to *MiMC* with respect to the relevant metrics in these applications.

2 Arithmetization

Zero-knowledge proof systems for arbitrary computations, multi-party computation, and indeed, even fully homomorphic encryption, share more than just a superficially similar characterizer of complexity. Underlying these advanced cryptographic protocols is something more fundamental: the protocols stipulate applying algebraic operations to mathematical objects, and somehow these operations correspond to computations.

This correspondence is not new. It was originally introduced by Razborov [37] as a mechanical method in the context of computational complexity and first applied to cryptographic protocols by Lund *et al.* [32]. This method, known as *arithmetization*, characterizes a computation as a sequence of natural arithmetic operations on finite field elements.

Arithmetization translates computational problems — such as determining whether a nondeterministic Turing machine halts in T steps — 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.

In the remainder of this section we survey three applications of arithmetization in cryptography: zero-knowledge proofs in the Turing or RAM models of computation, zero-knowledge proofs in the circuit model of computation, and multi-party computation. The purpose of this survey is to introduce the mechanics and to set the stage for analyzing efficiency and design techniques. These modalities of computation provide the reference frame according to which the rest of the paper proceeds.

The astute reader will notice that fully homomorphic encryption is frequently listed among the target applications of arithmetization-oriented ciphers and yet is missing from both the above discussion and the surveys below. The ciphers proposed in this paper rely heavily on a family of techniques we call *acausal computation*, in which the state of the system at the next computational step cannot be described as having been caused by the state at the previous step — see Section 3.1 for a more precise description of this term. To the best of our knowledge, fully homomorphic encryption does not presently admit acausal computations. As a result, our ciphers are ill-suited to this application scenario. We opt to restrict focus to context and aspects relevant for *Vision* and *Rescue*; while the design of ciphers for fully-homomorphic encryption shares much of the context surveyed in this paper and can properly be characterized as a subfield of arithmetization-oriented cipher design, we leave this particular question out of the scope of the present paper and to future work.

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.[6] 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 [10], we say that a ZK proof system for $\mathcal{L}$ is

- *scalable* if two conditions are satisfied simultaneously for all instances ℓ, $|\ell| = n$: (i) proving time scales quasi-linearly, like $\mathsf{poly}(n) + T(n) \cdot \mathsf{poly}\log T(n)$, and (ii) verification time scales like $\mathsf{poly}(n) + \mathsf{poly}\log T(n)$.

[6] 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.

– *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 to membership of ℓ in $\mathcal{L}$ by interacting with a prover who has a sufficiently high probability of convincing the verifier.

Argument systems that possess all of the properties above are referred to as ZK-STARKs, and have been recently implemented in practice [10], following theoretical constructions [11, 12] (cf. [9] 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 [10]. Both Turing machines and Random Access Memory (RAM) machines can be represented succinctly using AIRs that we describe briefly now, and more formally in Appendix D.[7]

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 C 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 $\mathcal{P}$. While the degree d does not affect the size of the AET, it does affect the execution time and the proof size.

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. Succinct computations can be "unrolled" into arithmetic circuits, and several compilers exist that achieve this, *e.g.*, [13, 36, 39]. 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

[7] 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.

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) [28] and Rank-One Constraint Satisfaction (R1CS) systems [13] 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 [28]. Additional transparent systems such as Ligero [4], BulletProofs [17] and Aurora [14] 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 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.

Using a linear sharing scheme, additions are essentially free and multiplication requires communication between the parties. The number of such multiplications 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 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.* [23]. 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 $[r^{-e}]$ 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 [6].

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 benchmarking. The key observation is that *some* polynomials with large powers *can* be efficiently computed over MPC — *even when counting the offline phase.*

3 Design Considerations and Concepts

Cipher design has been subject of research since the publication of the Data Encryption Standard (DES) [20]. Since then, science has progressed to the point where designing a new block cipher can be as simple as following a formula: choose a family of basic operations (*e.g.*, ARX) and a general structure (*e.g.*, (G)Feistel, or SPN), components with known cryptographic properties (*e.g.*, S-boxes with high non-linearity, a linear layer with fast diffusion), add constants to break symmetry, and set the number of rounds based on theoretical arguments (*e.g.*, the wide trail strategy) or using automatic tools (e.g., MILP and SAT-solvers [31,33,34]). This approach, if used properly, results in a secure algorithm that is resistant to known attacks.

In contrast, arithmetization-oriented ciphers necessitate different design considerations and cannot be understood in the context of traditional cipher design. While such a design formula could exist in principle, the field has not yet progressed to the point where it can be articulated. In this section, we highlight and discuss the considerations identified during our design process that are unique to arithmetization-oriented algorithms. This section is independent of our cipher designs. To the extent that it raises questions or concerns, these are answered and addressed in the context of *Marvellous* universe's designs of ciphers in Section 4.

3.1 Acausal Computation

In a procedural model of computation, the state of the system at any point in time can be uniquely determined as a simple and efficiently computable function of the system's state at the previous point in time. The arithmetic modalities of computation considered in this paper are capable of violating this procedural intuition. While all participants in the protocols are deterministic and procedural computers, some emergent phenomena are best interpreted either with respect to a different time axis or without any respect at all to the passage of time. From this perspective, they seem to undermine the constraining character of causality or violate it altogether. We call these phenomena *acausal computations.*

It is possible to design and define ciphers in terms of acausal computations. Doing so can offer security against particular or general attacks without having to increase the number of rounds. This benefits the efficiency of the advanced cryptographic protocol capable of computing the acausal operations efficiently. As a result of this design strategy, the cipher might be more expensive to evaluate on traditional, progressive computers; however, this is not the defining metric to begin with.

Consider for example the inversion operation $x \mapsto x^{q-2}$, for some $x \in \mathbb{F}_q$. When the field is large, so is the exponent, and as a result a progressive evaluation is expensive. We show how this operation is captured efficiently by acausal computation in various arithmetic modalities.

In the case of zero-knowledge proofs, the particular variant of acausal computation is known as *non-determinism*. The honest prover, who has evaluated the cipher locally, is in possession of all the intermediate states including x and $y = x^{q-2}$, and the verifier possesses only commitments to these values. The verifier is incapable of computing the values directly, but establishing that the expressions $x(1-xy)$ and $y(1-xy)$ both evaluate to 0 accomplishes the desired effect — convincing the verifier that y was computed correctly from x.

In the case of multi-party computations, the acausal computation originates from the capability of *masked operations* to offload certain calculations to the offline phase, where they do not affect online efficiency. In particular, in the offline phase the protocol prepares two shared values, $[a]$ and $[b]$ satisfying $a \neq b$ and $b = a^{-(q-2)}$. Then in the online phase, $[ax]$ is opened and the result is inverted locally before being multiplied with $[b]$, yielding $[y] = (ax)^{q-2}\,[b]$. This description ignores the special care necessary when $x = 0$ but the point remains that the number of online multiplications (which are expensive) is independent of the power to which the shared secrets are raised.

Another example of acausal computation arises in the polynomial modeling prelude to deploying a Gröbner basis attack — which is arguably another arithmetic modality of computation. The observation here is that the attack does not need to follow the same sequence of events involved in evaluating the cipher. The attacker can search for x and y simultaneously and require their consistency through the polynomial equation $xy - 1 = 0$. Note that the adversary may choose to ignore case $x = 0$ if the probability of this event is sufficiently small, as it is when working over large fields.

The takeaway is that acausal computation adds another dimension to cipher design, and opens the door to completely new types of constructions. While all algorithms presented in this paper (and, to the best of our knowledge, all published algorithms in this space) follow a traditional, progressive structure of an iterative cipher, acausal computation admits a departure from this strategy. Since progressive evaluation is no longer necessary, one can go a step further and consider operations whose progressive evaluation is expensive. The space of admissible ciphers extends even beyond functions to relations between pairs of objects, or of tuples even, whose defining computations admit an efficient acausal

representation. We leave the design of relational ciphers, along with a compelling selling point therefor, as an open question.

3.2 Efficiency Metrics

Unlike their traditional counterparts, arithmetization-oriented ciphers do not attempt to minimize execution time, circuit area, energy consumption, memory footprint, *etc.* — at least not as a first order consideration. Instead, these ciphers optimize algebraic complexity as described in terms of *AIR* or *R1CS* constraints for zero-knowledge proofs; and *number of multiplications*, *number of offline and online rounds of communication* for MPC. The common feature of these metrics is the gratuitous nature of linear operations. With respect to non-linear operations, each metric introduces its own subtleties. Even the cost of a single multiplication differs from metric to metric depending on where in the cipher that multiplication is located.

To illustrative this discrepancy, consider a state consisting of m field elements in some field $\mathbb{F}_q$. Suppose that we want to square one of these m elements over a non-binary field. This would require 1 multiplication in an MPC protocol, but would require an entire row (m entries) in algebraic execution trace of a STARK proof. Should we want to raise the element to a higher power α, we can use masking techniques in MPC at a fixed online cost that is independent of α, and yet it would require $\log_2(\alpha)$ R1CS constraints. The exception is when α has a small inverse in $\mathbb{Z}/(q-1)\mathbb{Z}$; then the R1CS representation can be optimized with an acausal computation.

At the risk of stating the obvious, even when restricting to zero-knowledge proof systems, ciphers can have a different cost depending on whether they are encoded as R1CS and AIR. For instance, raising a value to the power α requires $\log_2(\alpha)$ R1CS constraints, meaning that the cost is the same for all values in the range $]2^{\log_2(\alpha)-1}, 2^{\log_2(\alpha)}]$. In contrast, a system encoded in AIR can specify the maximal degree d of the polynomials describing the system, giving rise to a cost of $\log_d(\alpha)$ AIR constraints.

Another subtlety is introduced by the introduction of acausality, which is used precisely because its effect on the relevant efficiency metrics is small. In particular, low-degree power maps, low-degree affine polynomials, and their functional inverses, all have efficient acausal computations.

Importantly, and unlike in the case of traditional cipher design, the size of the field over which the cipher is defined, is immaterial to its cost of operation. For example, changing the base field of a hash function from $\mathbb{F}_{2^{128}}$ to $\mathbb{F}_{2^{256}}$ doubles the digest length at no additional cost.

The flipside of the cheapness of native field operations is the expensiveness of non-native operations that traditional ciphers typically are composed of. For example, the exclusive-or operation is extremely cheap for traditional ciphers because the platforms on which they run represent everything as sequences of bits; however, applying the same operation to elements of an odd-characteristic field requires first computing this bit expansion, which is prohibitively expensive.

Arithmetization-oriented ciphers must sacrifice the security benefits conferred by mixing algebras.

3.3 Cryptanalytic Focus

In traditional cipher design, statistical attacks — particularly, differential and linear cryptanalysis — are considered to be the main threat. The alternatives to statistical attacks, algebraic attacks, seldom deliver better results despite being a subject of active research. However, the opposite seems to be the case for arithmetization-oriented ciphers, for two reasons. First, the flexibility in choosing the field size, the gratuitous nature of scalar multiplication, and acausal computation, allow killing statistical attacks in a rather small number of rounds.

Second, and more importantly, the optimization of ciphers for arithmetic modalities of computation has the unfortunate side-effect of enabling attacks that exploit their low arithmetic complexity. Any cipher whose operations are described by simple polynomials gives rise to a range of attacks that manipulate those same polynomials algebraically (and enjoy the speedup afforded by acausal computation). While it is true that any function from finite fields to finite fields can be represented by a polynomial, the problem is that arithmetization-oriented ciphers make this polynomial representation concise and thereby reduce the complexity of algebraic attacks that are otherwise wildly infeasible. Among this class of algebraic attacks we count the interpolation attack [29] and the GCD attack [3, §4.2], and, warranting particularly close attention, Gröbner basis attacks. For an overview on the processes involved in Gröbner basis attacks, we refer the reader to Appendix A; we proceed here assuming familiarity with these concepts.

The interpolation and GCD attacks rely on the *univariate* polynomial expression of the ciphertext in function of the plaintext (or vice versa). Their complexity, and their countermeasures, are mostly understood. In essence, it is sufficient to ensure that the algebraic degree of the univariate polynomial describing the algorithm is of high enough degree and dense for the algorithm to be deemed secure against these attacks.

In contrast to these attacks, Gröbner basis attacks admit a *multivariate* polynomial description and are much more difficult to qualify in terms of complexity. This difficulty stems from a variety of sources:

- The field of arithmetization-oriented cipher design is a relatively new field, spurred by recent progress in advanced cryptographic protocols. For ciphers not optimized for arithmetic complexity, merely storing the multivariate polynomials in memory tends to be prohibitively expensive, let alone running a Gröbner basis algorithm on them. As a result, Gröbner basis attacks are rarely considered and poorly studied.[8]
- There may be many ways to encode a cipher as a system of multivariate polynomials, or more generally, to encode an attackable secret as the common

[8] Interestingly, AES, which is surprisingly arithmetizible considering that it was not designed as such, also admits certain algebraic attacks [18].

solution of a set of multivariate polynomial equations. As such, Gröbner basis attacks do not constitute one definite algorithm but a family of attacks whose members depend on the particular choices made while modeling the cipher as a collection of polynomials.

- The complexity of Gröbner basis algorithms is understood only for systems of polynomial equations satisfying a property called *regularity*, which corresponds to the algorithms' worst-case behavior. Even if a given system of polynomial equations is regular, it is difficult to prove that this is the case without actually running the algorithm. The complexity of Gröbner basis computation of irregular systems can be characterized in terms of the system's *degree of regularity*, but once again there is no straightforward way to compute this degree without actually running the Gröbner basis algorithm.
- In some cases, the actual Gröbner basis calculation is relatively simple but the corresponding variety contains parasitical solutions in the field closure. Additional steps are then required to extract the correct base field solution, and these post-processing steps may be prohibitively complex. The parasitical solutions are typically eliminated by converting the Gröbner basis into one with a lexicographic monomial order, at which point at least one basis polynomial is univariate; factorizing this polynomial identifies the solutions in the base field. The complexity of monomial order conversion can be, and often is, captured via that of the FGLM algorithm [26]; however an alternative algorithm called Gröbner Walk does not have a rigorous complexity analysis and yet is observed to outperform FGLM sometimes in practice [16]. More fundamentally, the diverse range of options afforded to the attacker when modeling the cipher in terms of polynomials as well as during other steps of the Gröbner basis attack, suggest that this typical strategy of monomial order conversion and factorization may be merely one out of many: it is eminently plausible that there are alternative strategies to filter out parasitical extension field solutions.

The dual design criteria of both having an efficient arithmetization and offering security against Gröbner basis attacks seem to be fundamentally at odds with each other. A concise polynomial description of a cipher benefits both the algebraic attack and the advanced cryptographic protocol that uses it. Consequently, the question of security against Gröbner basis attacks is *the* crucial concern raised by arithmetization-oriented ciphers, and no such proposal is complete without explicitly addressing it.

We observe that non-deterministic encodings used in zero-knowledge proofs have a counterpart in the cipher's polynomial modeling and make both the zero-knowledge proof and the Gröbner basis algorithm more efficient. Furthermore, we conjecture that this duality is necessarily the case, even for tricks and techniques that we may have overlooked.

The relative importance of Gröbner basis attacks is illustrated by *Jarvis* [5] and *MiMC* [3], two arithmetization-oriented ciphers that were proposed with explicit consideration for a wide range of attacks, but not attacks based on computing Gröbner bases. However, shortly after its publication, a Gröbner basis

attack that requires only a single plaintext-ciphertext pair was used to discover non-ideal properties in *Jarvis* [2]. An investigation of *MiMC* using the same attack was argued to be infeasible [2, Sec. 6]. While finding the Gröbner basis is easy, the next two steps — monomial order conversion and factorization of the resulting univariate polynomial — are not, owing to the infeasibly large number of parasitical solutions in the field closure.

However, relying on the large number of parasitical solutions for security against Gröbner basis attacks is a new security argument and a risky one. The simple observation that using more than just one plaintext-ciphertext pair makes the system of equations overdetermined, and thus filters out all parasitical extension field solutions with overwhelming probability, seems to undermine this argument. We note that the complexity analysis of overdetermined polynomial system solving requires delicate attention and it is conceivable that the resulting attack is *also* infeasible but for a different reason. However, the point is that even if this is the case, *MiMC*'s security is not guaranteed by the large number of parasitical solutions. Either way, these observations raise the question whether there is a systematic argument for Gröbner basis security that does not depend on the particular flavor of the attack. In Section 4.5 we answer this question positively by providing such an argument.

3.4 Translation of Existing Cryptographic Constructions

Cryptographic primitives are generally not used directly but as part of a larger scheme (*e.g.*, a mode of operation). When using an arithmetization-oriented primitive as part of such a scheme, it is important to address several concerns, starting with efficiency. Consider for example AES-CTR — it is easy to see that this mode of operation mixes two algebras: $\mathbb{F}_{2^8}$ for the block cipher part and $\mathbb{Z}$ for the counter. The result is a mode of operation with a prohibitively inefficient arithmetization.

Another important aspect that is easy to overlook is that the interface of the scheme may not be properly defined to work with field elements. As an instructive example we consider sponge constructions. A sponge construction generates a hash function from an underlying permutation by iteratively applying it to a large state. The state of a sponge function is defined to consist of $b = r + c$ **bits**, where r and c are called the *rate* and the *capacity* of the sponge, respectively. In other words, sponge functions are inherently defined to work over vector spaces of $\mathbb{F}_2$ (with the exclusive-or and conjunction operations as their native operations). We show in Section 4.4 how to fix this mismatch of algebras specifically for the case of sponges; however, such a straightforward fix might not always be possible.

Perhaps most importantly is to note that even if the security of a construction is well understood in the traditional setting, this knowledge may not be transferable to the arithmetized variant. A case in support of this point is the Merkle-Damgård construction in the face of Gröbner basis attacks. We observe that in certain cases the degree of regularity grows slowly but surely as a function of the round number when Davies-Meyer is used, whereas the degree of regularity remains constant(!) when Miyaguchi-Preneel is used. This is a surprising

observation since PGV hash functions are believed to be interchangeable in practice. We conjecture that absence of growth is due to the interface through which the unknowns are introduced. In particular, in Miyaguchi-Preneel the chaining value is introduced via the key interface and since this value is initially known, the key schedule does not contribute to the complexity of the resulting system of equations. We suspect that this vulnerability has an analogue in traditional cryptanalysis. We note that the practical interchangeability of the PGV constructions remains an open question.

The key takeaway from this section is that using schemes that are well understood in the traditional model may not be straightforward in the arithmetic model. Before instantiating a primitive in an existing construction, it is important to check that the construction is efficient with respect to the application at hand, properly defined, and that its security proof translates to the computational model being considered.

3.5 Concluding Words

Our survey of the advanced cryptographic protocols employing arithmetic modalities of computation is by no means complete. Consequently, our matching survey of the design considerations induced by the advanced cryptographic protocols that we do cover, is likewise incomplete.

For example, fully homomorphic encryption is missing from our list of cryptographic protocols and yet induces other design considerations, starting with the conjectured unavailability of acausal operations. Another difference is that multiparty computations and zero-knowledge proofs tend to be flexible with respect to the operating field, and in principle this field is not an input to the security calculation. By contrast, the field of choice in fully-homomorphic encryption is intricately linked to the security provided by the encryption scheme. A third difference with fully homomorphic encryption is that both additions and multiplications accrue noise, although the noise increase due to multiplication is much greater. As a result, additions are not free but merely much cheaper than multiplications.

It is possible that we overlooked other advanced cryptographic protocols employing arithmetic modalities of computation, or that other are yet to be invented. If there is a demand on the part of these protocols for symmetric ciphers, then the design considerations for such ciphers ought to be re-evaluated in light of the target protocol and application. In such an event, the points and questions raised by our analysis provide an ample roadmap for such a reassessment.

Lastly, we note that the field of algebraic attacks seems rather underexplored. As a result, it is difficult to make a compelling security argument valid for the entire family of attacks. We expect third party analysis to contribute to fleshing out this field and hope that this analysis confirms the merit of our design principle for addressing algebraic attacks (Section 4.5).

4 Design Decisions

Following the discussion in Section 3 it is clear that designing an arithmetization-oriented cipher is different from designing a traditional one and that the different considerations lead to different design decisions. In this section we explain and motivate the design decisions we made in the course of developing the *Marvellous* designs *Vision* (Section 5) and *Rescue* (Section 6). We begin by explaining our motivating principles.

4.1 General Structure

Vision and *Rescue* are two primitives based on substitution-permutation networks operating over fields of even and odd order, respectively. Both families manipulate a state of $m > 1$ elements seen as a column vector. A round of the function includes two steps. In every step an S-box is applied to each of the m state elements, followed by an MDS matrix which mixes the elements together.

The S-box consists of a power map, possibly composed with an affine layer. The exact maps used are detailed in the parts specific to *Vision* and *Rescue* and so is the difference between the the odd and even step within each round. The cipher is an iterative application of the round function N times with different round keys derived from the key schedule.

4.2 Key Schedule

The key schedule of the algorithms reuses the round function. The master key is fed through the plaintext interface and random round constants are used where the subkey is normally injected. The subkeys are then determined as the value of the state immediately following the constant injection.

The round constants are derived in the following way: we use SHAKE256 to expand a short seed into enough randomness from which one samples the first round constant (with rejection as necessary to ensure that the bit string does not represent a subfield element nor an integer larger than or equal to the prime modulus). All subsequent constants are obtained by applying an affine transformation to the previous one. The first round constant and the coefficients of the affine transformation can be generated deterministically using the code provided in [38].

In recent years, driven by the advent of lightweight cryptography, complex key schedules have fallen out of favor. For the *Marvellous* designs we have decided to take the opposite approach, namely a heavy (*i.e.*, non-linear) key schedule. This complexity is motivated by the following arguments:

– The domain of arithmetization-oriented ciphers is relatively new and it pays to err on the side of safety until the landscape of possible attacks has been explored more thoroughly.

– One of the use cases of arithmetization-oriented ciphers is hashing; and in this case it is possible to completely hide the complexity overhead of the key schedule as its input is a known IV or fixed key. In other cases it may be possible to amortize the cost of the key schedule over the cost of the entire execution.
– A straightforward Gröbner basis attack on the block cipher represents a key recovery from one or a few plaintext-ciphertext pairs. When the key schedule is simple — say, linear — then the same variables that are used to represent the key in one round can be reused across all other rounds. A complex key schedule introduces many more variables and equations, making the system of equations that much more difficult to solve. Reusing the round function in the key schedule is a conceptually simple way to require at least as many polynomials and variables in the polynomial modeling step as are required to attack the hash function.
– A less straightforward Gröbner basis attack on the block cipher targets the injected subkeys rather than the master key. However, as these are different and have no attackable relation, they must be treated as independent variables. Consequently, at least $2N$ plaintext-ciphertext pairs (one per step) are necessary to uniquely determine these subkeys. With the resulting explosion in the number of variables and equations, even a very mild degree of regularity makes the system of equations unsolvable in practice.

4.3 Efficiency

Throughout the *Marvellous* design, we only use arithmetization-efficient maps or the functional inverse of one. The realization of functional inverse maps is not always efficient when implemented straightforwardly. However, all advanced protocols considered in this paper enable acausal computations that make the functional inverse of an arithmetization-efficient operation itself arithmetization-efficient as well.

A particularly useful property of our designs is the inverse trade-off between m and N (*i.e.*, the number of field elements in the state and the number of rounds, respectively). We see that for higher m, the degree of regularity grows faster in each round. This allows to treat m as a parameter that can be tweaked in order to favor a lower multiplicative depth in exchange for a lower base field size. For example, a large m can be used to build an n-ary Merkle-tree rather than a binary one and thus shrink the authentication paths. As the S-boxes operate in parallel, a large m allows to compress m multiplications into a single communication round in an MPC protocol.

4.4 Arithmetic Sponge

A sponge construction generates a hash function from an underlying permutation by iteratively applying it to a large state [15]. Traditionally, the state is thought of as consisting of $b = r + c$ bits, where r and c are called the *rate* and the *capacity* of the sponge, respectively. In every iteration of the absorbing phase,

r bits of the input are injected into the state until there are no new input bits left; in every iteration of the squeezing phase, r bits of the state are read out until the desired output length is met. We slightly adapt this definition to allow for hashing of field elements. Instead of working over bits, the rate part now consists of r_q field elements in $\mathbb{F}_q$. The remaining $c_q = m - r_q$ elements of the state constitute the capacity and their size determines the security of the sponge.

To turn the block ciphers into permutations, the secret key is fixed to zero. The resulting permutation is then used in a sponge construction to obtain an extendable output function and, if the output length is fixed, a hash function. 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.

The resulting sponge absorbs (using field addition) and squeezes r_q field elements per iteration and offers $\log_2(c_q)$ bits of security. Note that increasing r_q and keeping c_q fixed increases the throughput of the sponge without significantly affecting the cost and not at all the security.

4.5 Security

We now give a high-level overview of the algorithms' security countermeasures applied to inoculate them against attacks. A more rigorous description of how each attack is prevented can be found in Appendices G–H.

Statistical Attacks In the design of traditional symmetric-key primitive, resistance to differential and linear cryptanalysis is of utmost importance. Building on the work of Nyberg [35] we see that S-boxes consisting of a power map have good differential and linear properties and that these properties can be easily derived once the field is specified. Using the wide trail strategy we bound the maximum differential probability and linear correlation of an active S-box.

An interesting observation here is that these quantities improve directly (from the designer's point of view) as a function of the field size. Since the state is treated as a column vector (rather than a matrix as in AES) fast diffusion is also achieved since the MDS guarantees that if at least one S-box is active before the MDS, $m + 1$ S-boxes would be active afterwards.

We require that the field $\mathbb{F}_q$ is at least 4-bit wide and that $2m \geq q$ such that a state-wide MDS matrix exists, which offers resistance against differential and linear cryptanalysis after four rounds. We refer the interested reader to [22] for a more elaborate description of the wide trail strategy, and to Appendices G–H for the exact derivation in our case.

It is unclear how linear cryptanalysis would look like for ciphers operating over elements of $\mathbb{F}_p$ with p prime. Normally, linear cryptanalysis searches for a linear combination of input-, output-, and key bits that is unbalanced, *i.e.*, biased towards 0 or towards 1. As such, linear cryptanalysis seems tailored to work over the field $\mathbb{F}_2$. No analogue to this behavior exists for $\mathbb{F}_p$. All reasonable extensions of linear cryptanalysis to this case would treat an element in $\mathbb{F}_p$ as we would normally treat bits and search for a expression approximating a ciphertext

(which is an element in $\mathbb{F}_p^m$) as a multivariate linear polynomial in $\mathbb{F}_p$. Since the polynomial is dense and of high degree, there is no straightforward way to go about finding such a linear approximation. However, we stress that we do not have a rigorous argument for the inapplicability of linear cryptanalysis in this setting and the dual questions — how to lift linear cryptanalysis to this setting, and how many rounds can be attacked — remains open.

Structural Attacks Self-similarity attacks work by splitting a cipher into multiple sub-ciphers that are similar to one another, for some definition of similarity. This allows to attack one of the sub-ciphers and use the self-similarity to cleverly link this part with the other ones.

The invariant subspace attack works by observing that all the values of a certain subspace that enter the round function are mapped to outputs within a subspace. The input and output subspaces do not have to be the same.

The standard way to resist these attacks is to inject round constants which break the similarity between different parts of the algorithms, and lift the state out of the subspace being attacked. In our case, we opt to add these constants to the key schedule, resulting in a "fresh" value injected into the state in every round. Note that even when instantiated inside a sponge function, the key schedule still outputs a non-zero value in each step and that value is injected into the state. The improved efficiency in this case is due to the fact that the constants can be precomputed and hard-coded into the realization of the algorithm.

Algebraic Attacks The way to resist most algebraic attacks is to ensure that the polynomial describing the output of the primitive is dense and of high degree. Initially, having a high degree polynomial appears to be at odds with having an efficient arithmetization-oriented primitive. We resolve this apparent contradiction by using maps of high degree that can be computed efficiently owing to acausal operations and operating on $m \geq 2$ state elements. The MDS matrix thus ensures a good mixing, resulting in a dense polynomial.

Gröbner basis attacks are of particular interest here. Recall that a Gröbner basis attack consists of the following steps:

(i) computing the Gröbner basis in *degrevlex* order;
(ii) converting the Gröbner basis into *lex* order;
(iii) factorizing the univariate polynomial, and back-substituting its roots.

We use the following design principle to argue security against Gröbner basis attacks:

the security of arithmetization-oriented ciphers against Gröbner basis attacks should come from the infeasible complexity of computing the Gröbner basis in degrevlex order.

This principle guarantees that the Gröbner security is independent of the presence of parasitical solutions in the field closure; if present and large in number,

these parasitical solutions represent a superfluous security argument because the attacker has to get past step (i) in order to get to step (iii). More importantly, with this principle, the number of parasitical extension field solutions required for an infeasible univariate factorization is no longer a constraining factor in determining of the number of rounds.

In order to guarantee that finding the first Gröbner basis is prohibitively expensive, we implement the cipher and an attack and observe the degree of regularity experimentally for small round numbers. We assume a constant relation between the observed concrete degree of regularity, and the degree of regularity of a regular system of the same number of equations, degrees, and variables. Conservatively, assuming $\omega = 2$ as the linear algebra constant and extrapolating from there, we set the number of rounds such that this Gröbner basis attack has the required complexity.

This Gröbner basis attack represents a preimage search attacking the sponge-based hashing mode as described in Section 4.4 with $m = 2$, in which one data block is absorbed and one digest block is squeezed out. Alternative Gröbner basis attacks induce a greater complexity due to the increase in the number of variables without a disproportionate increase in the number of equations. When used as a block cipher instead of as a hash function, variables and equations need to be introduced to account for the key schedule. Since this key schedule is as complex as the sponge-based hash function, the resulting Gröbner basis attack must be at least as expensive. These attacks on *Vision* and *Rescue* are discussed in more detail in Appendices G–H.

Number of Rounds To set the number of rounds for each primitive we consider ℓ, the maximal number of rounds that can be attacked by any of the attacks above. Our analysis shows that algebraic attacks other than Gröbner basis ones do not extend beyond three rounds, and that statistical attacks can be used against four rounds at most. For most parameters we considered, the most dangerous attack is the Gröbner basis attack and we discuss its analysis when determining the number of rounds in the respective sections. Having determined ℓ, we set the number of rounds to be 2ℓ with a minimum of 10 rounds.

4.6 Concluding Words

Given the present state of development of arithmetization-oriented cipher design, our design choices can hardly be argued to be the right ones or to enjoy widespread consensus as being good ones. However, the common theme throughout all choices is the preference for erring on the side of safety, thereby minimizing the risk of unforeseen fatal attacks. As we see in the sequel, even with this conservative approach, our ciphers are extremely efficient in the use cases we identified. Still, we hope that independent third-party analysis reaches the conclusion that our design choices were indeed too conservative, and that the complexity and security margins can safely be reduced.

5 *Vision*

We now describe the first family of the *Marvellous* universe of ciphers, *Vision*, whose design is inspired by that of AES. Since we already discussed the design decisions leading to *Vision* in Section 4, we only discuss here the technical specification of the algorithm.

The State The native field in which *Vision* operates is $\mathbb{F}_{2^{n/m}}$ where n is the security level desired (in bits) and m is the number of field elements in the state. The state is viewed as a column vector of m field elements and is an element of the vector space $\mathbb{F}_{2^{n/m}}^m$.

S-Box The S-box of *Vision* consists of two operations composed with one another. The first is the inverse power map which is expressed as the following power map

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

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 affine polynomial is of the form

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

Such a polynomial is a permutation over $\mathbb{F}_{2^{n/m}}$ if and only if its linear part only has the root 0 in $\mathbb{F}_{2^{n/m}}$.

In fact, two S-boxes (π_1, π_2) are used in *Vision*. The first S-box, π_1, consists of the inversion function composed with the functional inverse of the $\mathbb{F}_2$-affine polynomial. The second S-box, π_2, uses the multiplicative inversion composed with a direct evaluation of the same polynomial. When evaluated in the forward direction (*i.e.*, in encryption mode), π_1 ensures that the algebraic degree of the polynomial description of the algorithm is sufficiently high. Similarly, when evaluated backwards (*i.e.*, in decryption mode), π_2 achieves the same goal.

Round Function The round function consists of two steps. In each step, the state goes through a non-linear layer followed by a multiplication with an MDS matrix. The non-linear layer applies π_1 to each of the m elements if this is the first step of the round, and applies π_2 if it is the second. Followed by the non-linear step, an MDS matrix which is the same for both steps is used to mix the

state. A schematic description of a single round (two steps) of *Vision* is depicted in Figure 1 and the pseudo-code of the cipher is listed in Algorithm 1.

Algorithm 1: *Vision*

Input: Plaintext P, round keys K_s for $0 \leq s \leq 2N$
Output: *Vision* (K, P)

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

To generate the ciphertext from a given plaintext, the round function is iterated N times with a key injection before the first round, between every two steps, and after the last round.

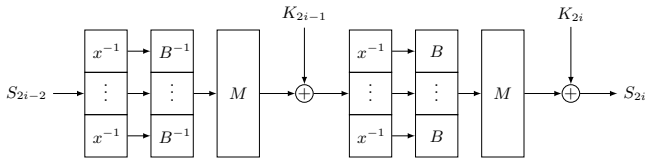


Figure 1: A single round (two steps) of *Vision*

Choosing the Number of Rounds As explained in Section 4, the number of rounds is determined by the Gröbner basis attack. Experiments on reduced parameters show that the base-2 logarithm of the complexity of such an attack is lower-bounded by $5.5mN$. 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.

6 *Rescue*

The second family of algorithms in the *Marvellous* universe is *Rescue*. *Rescue* is similar to *Vision*, but this time operating on elements of prime fields rather than binary ones.

The State The native field in which *Rescue* operates is $\mathbb{F}_p$. The state is viewed as a column vector of m field elements and is seen as an element of the vector space $\mathbb{F}_p^m$. The security level afforded by the algorithm is $m \cdot \log_2(p)$.

S-Box Similar to *Vision*, *Rescue* uses a pair of S-boxes π_1 and π_2. The S-boxes consist of the power maps $x^{1/\alpha}$ and x^α, respectively, where α is the smallest prime such that $\gcd(p-1, \alpha) = 1$.

For most fields, $\alpha = 3$ suffices. When possible we recommend to choose the field such that $\alpha = 3$ is viable. In some cases the field is determined by the intended application and cannot be chosen freely. For example, the 255-bit prime field $\mathbb{F}_r$, which is used for the multiplications made over the BLS12-381 curve used by ZCash, does not satisfy $\gcd(r-1, 3) = 1$ making $\alpha = 3$ unsuitable for this case. Instead, to use this field one can choose $\alpha = 5$ since $\gcd(r-1, 5) = 1$.

The map $x^{1/\alpha}$ 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 $\gcd(p-1, \alpha) = 1$. More specifically, $1/\alpha \cdot \alpha \equiv 1 \pmod{p-1}$.

Round Function The round function of *Rescue* consists of two steps. In the first step, π_1 is used, followed by and MDS matrix. In the second step, π_2 is used, again followed by an MDS matrix.

To generate the ciphertext from a given plaintext, the round function is iterated N times with a key injection before the first round, between each two steps, and after the last round.

A schematic description of a single round (two steps) of *Rescue* can be found in Figure 2 and the pseudo-code of the cipher is listed in 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 .

Algorithm 2: *Rescue*

Input: Plaintext P, round keys K_s for $0 \leq s \leq 2N$
Output: *Rescue* (K, P)

```
State_0 = P + K_0
for r = 1 to N do
    for i = 1 to m do
        Inter_r[i] = Σ_{j=1}^{m} M[i,j](State_{r-1}[j])^{1/α} + K_{2r-1}[i]
    end
    for i = 1 to m do
        State_r[i] = Σ_{j=1}^{m} M[i,j](Inter_r[i])^α + K_{2r}[i]
    end
end
return State_N
```

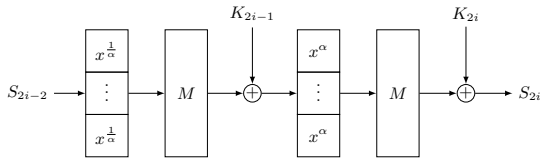


Figure 2: One round (two steps) of *Rescue* where the addition with the key is taken over a prime field.

Choosing the Number of Rounds Similar to the case *Vision* we see that the most prominent attack is the Gröbner basis attack. We find that the base-2 logarithm of the attack complexity is lower-bounded by $4mN$. Accounting for a factor two security margin, we set the number of rounds to $2\lceil \log_2(p)/4 \rceil$ with a minimum of 10 rounds, for a security level of $m \log_2(p)$.

7 Benchmarks

In this section we analyze the efficiency of *Vision* and *Rescue* with respect to three use cases: AIR constraints for ZK-STARKs (Section 7.1), Zero-Knowledge Proofs based on R1CS Systems (Section 7.2, and MPC protocols (Section 7.3). Section 7.4 provides a comparison of the algorithms with *MiMC-q/q* and *MiMC-2p/p*.

Notation. We use the following conventions. Variables of 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.*, $\{a_i, b_i \,|\, i \in [2]\} = \{a_1, a_2, b_1, b_2\}$.

7.1 AIR Constraints for ZK-STARKs

We begin by realizing the two algorithms in AIR, the Domain-Specific Language (DSL) used to encode ZK-STARKs. For the sake of readers not versed in the relevant definitions related to STARKs [10] we recall those, along with a simple motivating example in Appendices C and D.

Encoding of a *Vision* Step as a Set of AIR Constraints We present an AIR with $\mathsf{w} = 4m$, $\mathsf{t} = 2$ and degree $\mathsf{d} = 2$ 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 ($\mathsf{w} = 2m$) 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 $2m$ additional

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 C. 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.

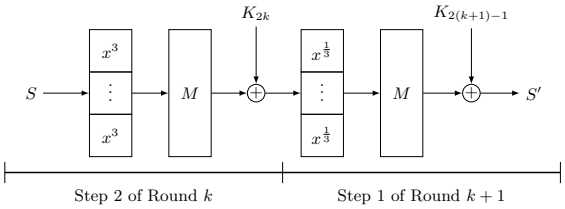


Figure 3: An adapted representation of a round of *Rescue* better suited for STARK evaluation.

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 or state function 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.

The representation of the *Rescue* round function admits an optimization owing to acausal computation. Consider an adapted round as shown in Figure 3. Here, the first step of the adapted round is "folded" into its second step. This leaves the first and last steps of the entire primitive to be taken separately. 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 $S[1], \ldots, S[m]$, $K_{2k-1}[1], \ldots, K_{2k-1}[m]$ and $K_{2(k+1)-1}[1], \ldots, K_{2(k+1)-1}[m]$,

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

Where we used that $K_{2k}[i] = \sum_{j=1}^{m} M[i,j]K_{2k-1}[i]^3$ for all $i \in [m]$. 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. Since the above encoding does not require K_{2k}, the key schedule admits a similar optimization. As a result, the *Rescue* key schedule AIR also has degree $\mathsf{d} = 3$, state width $\mathsf{w} = m$ and $\mathsf{t} = 1$ cycle per round. When the cipher 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*.

7.2 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 [36], ZK-SNARK [13], Aurora [14], Ligero [4], and Bulletproofs [17].

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 7.1), 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 $\alpha_j^2 = M[i,j]b_3$ .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.).

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.

7.3 MPC with Masked Operations

In this section we explore how to implement *Vision* and *Rescue* over MPC using masked operations.[9] We consider three masked operation techniques: one technique to find the inverse of a shared field element due to Bar-Ilan and Beaver [6]; one technique to raise a shared element to an arbitrary but known power due to Damgård *et al.* [23]; and one new technique to compute the compositional inverse of a low-degree linearized polynomial. The last two techniques are novel and their descriptions can be found in Appendix E.

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 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. Our MPC implementations of *Rescue* and *Vision* are agnostic of the particular zero-test as well as of the secret sharing mechanism. In the sequel we present figures without taking the zero test into account.

Computing a *Vision* Round over MPC 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.

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 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.

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

To evaluate the inversion step, we use the technique due to Bar-Ilan and Beaver [6], of which pseudocode is given in Appendix E. This procedure 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.

A similar approach can be used to compute $B^{-1}(x)$. To the best of our knowledge, this masking technique is novel and is thus an independent contribution of this paper. However, in the interest of brevity, we only describe it in Appendix E.2 together with pseudo-code.

The implementation of a round of *Vision* follows straightforwardly from using these building blocks, along with linear (and thus local) operations. 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 the m elements, they are performed independently and are hence parallelizable. The total complexity of *Vision* is therefore

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

Computing a *Rescue* Round over MPC 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.* [23]. 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 in Appendix E.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

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

7.4 Comparison

To compare *MiMC* with *Vision* and *Rescue*, we set $m = 2$, $n = 128$, $p = 2^{64}+13$, $q = 2^{129} - 45$ 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 stress that in the interest of a fair comparison with *MiMC*, we provide figures of merit only for the case of $m = 2$, which is on the one hand, the smallest m we deem secure, and on the other hand, the largest m that allows for such comparison. However, noting that our designs achieve an inverse trade-off between m and the number of rounds, setting m to a higher value would show that the *Marvellous* designs are even more efficient than how they are portrayed here.

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*. Since the absorption and squeezing of inputs and outputs in the case of *MiMCHash-q/q* are not native operations to the working field, they 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. 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 (*i.e.*, arithmetic complexity).

Table 1: Comparison of *Vision*, *Rescue*, *MiMC-q/q* and *MiMC-2p/p* over AIR. With $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{129} - 45$. We see that when evaluated as a block cipher, *MiMC* significantly outperforms both *Vision* and *Rescue*. When evaluated as a hash function *Rescue* outperforms both *MiMC* and *Vision*.

	Mode	Degree (d)	Width (w)	Cycles (t)	$\mathsf{w} \cdot \mathsf{t}$
Vision	BC	2	8	96	768
Rescue	BC	3	4	33	132
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

8 Conclusion

This paper explores the design of secure and efficient symmetric-key primitives for advanced cryptographic protocols based on arithmetization. It starts by surveying three protocols that fit this description — zero-knowledge proofs for the

Table 2: Comparison of *Vision*, *Rescue*, *MiMC-q/q* and *MiMC-2p/p* over R1CS. With $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{129} - 45$. We see that when evaluated as a block cipher, *MiMC* significantly outperforms both *Vision* and *Rescue*. When evaluated as a hash function *Rescue* outperforms both *MiMC* and *Vision*.

	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 $m = 2$, $n = 128$, $p = 2^{64} + 13$, $\alpha = 3$, $q = 2^{129} - 45$. We see that when comparing the number of multiplications, *MiMC* has the lowest count as a block cipher, with *Vision* as close second. For the hash case, it is *Vision* which has the lowest multiplication count. When considering the number of online rounds, which determines the execution time of the protocol, *Rescue* beats the other two algorithms both as a block cipher and as a hash function.

	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

Turing or RAM models of computation, for the circuit model, and multi-party computations.

The design considerations unique to designing arithmetization-oriented primitives are discussed. We show that the set of efficient building blocks available to the designer of an arithmetization-oriented cipher is different from those available to designers of a traditional cipher. We also observe that the efficiency metrics are different, as well as the security concerns. This last point we discuss at length, particularly in the context of Gröbner basis attacks, and we propose a strategy to ensure that an arithmetization-oriented cipher is secure against this class of attacks.

After this discussion we turn to designing two new families of arithmetization-oriented ciphers — *Vision* and *Rescue* — new members of the *Marvellous* universe. These primitives are benchmarked 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), and we compare them in these settings to *MiMC*. We see that the *Marvellous* algorithms are extremely efficient. Despite the conservative nature of these designs, they outperform *MiMC* in all but a few cases.

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., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic cryptanalysis of stark-friendly designs: Application to marvellous and mimc. Cryptology ePrint Archive, Report 2019/419 (2019), `https://eprint.iacr.org/2019/419`
3. 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
4. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: ACM - CCS 2017 (October 2017)
5. Ashur, T., Dhooghe, S.: Marvellous: a stark-friendly family of cryptographic primitives. IACR Cryptology ePrint Archive **2018**, 1098 (2018)

6. 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
7. Bardet, M., Faugère, J.C., Salvy, B.: On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In: Proceedings of the International Conference on Polynomial System Solving. pp. 71–74 (2004)
8. Bardet, M., Faugère, J., Salvy, B.: On the complexity of the F5 gröbner basis algorithm. J. Symb. Comput. **70**, 49–70 (2015). https://doi.org/10.1016/j.jsc.2014.09.025
9. 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 (3). Lecture Notes in Computer Science, vol. 10212, pp. 551–579 (2017)
10. 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`
11. 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)
12. 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)
13. 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)
14. 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`
15. 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
16. Buchmann, J.A., Pyshkin, A., Weinmann, R.: Block ciphers sensitive to gröbner basis attacks. In: Pointcheval, D. (ed.) CT-RSA 2006. Lecture Notes in Computer Science, vol. 3860, pp. 313–331. Springer (2006). https://doi.org/10.1007/11605805_20
17. 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`
18. Cid, C., Murphy, S., Robshaw, M.J.B.: Algebraic aspects of the advanced encryption standard. Springer (2006)
19. Collart, S., Kalkbrener, M., Mall, D.: Converting bases with the gröbner walk. J. Symb. Comput. **24**(3/4), 465–469 (1997). https://doi.org/10.1006/jsco.1996.0145
20. Coppersmith, D.: The data encryption standard (DES) and its strength against attacks. IBM Journal of Research and Development **38**(3), 243–250 (1994)
21. Cox, D.A., Little, J., O'Shea, D.: Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.). Undergraduate texts in mathematics, Springer (1997)
22. 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

23. 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)
24. 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
25. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (f_5). In: ISSAC 2002. pp. 75–83. ACM (2002)
26. Faugère, J., Gianni, P.M., Lazard, D., Mora, T.: Efficient computation of zero-dimensional gröbner bases by change of ordering. J. Symb. Comput. **16**(4), 329–344 (1993). https://doi.org/10.1006/jsco.1993.1051
27. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of pure and applied algebra **139**(1-3), 61–88 (1999)
28. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 626–645. Springer (2013)
29. 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
30. Knudsen, L.R.: Truncated and higher order differentials. In: FSE. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994)
31. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 9215, pp. 161–185. Springer (2015)
32. 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)
33. Mouha, N., Preneel, B.: Towards finding optimal differential characteristics for arx: Application to salsa20. Cryptology ePrint Archive, Report 2013/328 (2013), `https://eprint.iacr.org/2013/328`
34. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Inscrypt. Lecture Notes in Computer Science, vol. 7537, pp. 57–76. Springer (2011)
35. 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
36. 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)
37. 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)
38. Szepieniec, A., Dhooghe, S.: Marvellous (instance generator) (2019), `https://github.com/KULeuven-COSIC/Marvellous.git`
39. 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)
40. Wiedemann, D.: Solving sparse linear equations over finite fields. IEEE transactions on information theory **32**(1), 54–62 (1986)
41. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: Karloff, H.J., Pitassi, T. (eds.) STOC 2012. pp. 887–898. ACM (2012). https://doi.org/10.1145/2213977.2214056

A Gröbner Basis Attacks

We recall here some basic facts about attacking symmetric primitives using Gröbner basis algorithms. For more general information on the underlying mathematics, we refer the reader to Cox *et al.* [21]. For a specific description of the steps involved in attacking block ciphers with Gröbner bases, we refer to the excellent summary by Buchmann *et al.* [16].

An *ideal* $\mathcal{I} \subseteq \mathbb{F}_q[\mathbf{x}] = \mathbb{F}_q[x_1, \ldots, x_n]$ is the *algebraic span* of a list of polynomials $\{b_1(\mathbf{x}), \ldots, b_m(\mathbf{x})\}$, meaning that every member $f(\mathbf{x}) \in \mathcal{I}$ can be expressed as a weighted sum of the basis elements with coefficients taken from the polynomial ring: $f(\mathbf{x}) \in \mathcal{I} \Leftrightarrow \exists c_1, \ldots, c_n \in \mathbb{F}_q[\mathbf{x}] : \sum_{i=1}^{n} c_i(\mathbf{x}) \cdot b_i(\mathbf{x}) = f(\mathbf{x})$. An ideal can be spanned by many different bases; among these, *Gröbner bases* are particularly useful for computational tasks such as deciding membership, equality, or consistency. The task we are interested in is *polynomial system solving*: computing the ideal's *variety*, or the set of common solutions when equating all ideal members to zero.

A *monomial order* is a rule according to which to order a polynomial's terms. This rule is not just a convenience for mathematicians to read and write polynomials; it also affects how the polynomials are stored on a computer as well as the complexity of various operations on ideals. In general, the calculation of a Gröbner basis is fastest with respect to *degree reverse lexicographical ((de)grevlex)* order. However, whenever the variety contains a substantial number of solutions, a Gröbner basis in *lexicographic (lex)* order is preferable. A Gröbner basis in *lex* order guarantees the presence of at least one univariate basis polynomial. Factoring this polynomial and back-substituting its roots generates another, simpler, Gröbner basis again in *lex* order; iterative back-substitution produces all solutions. The FGLM [26] and Gröbner Walk [19] algorithms transform a Gröbner basis for one monomial order into one for another order.

The focus on degrevlex order for computing the first Gröbner basis owes in large part to the success of the celebrated F_4 and F_5 algorithms [25, 27]. In every iteration, these algorithms extend the working set of polynomials via multiplication by monomials to a certain *step degree*, before reducing the extended polynomials using linear algebra techniques — essentially Gaussian elimination on the Macaulay matrix. The F_5 algorithm stands out in this regard, because in this case it can be proven not to terminate before the step degree reaches the ideal's *degree of regularity* [7, 8], which is informally equal to the degree of the Gröbner basis in a degree-refining order such as *degrevlex* (but not *lex*). If a system of polynomial equations $\{f_i(\mathbf{x}) = 0\}_i$ is *regular* — exhibiting no non-trivial algebraic dependencies in the same sense that non-singular matrices exhibit no linear dependencies — then the degree of regularity is given by the Macaulay bound: $d_{\text{reg}} \leq 1 + \sum_{i=1}^{m}(\mathsf{deg}(f_i) - 1)$.

When there are more equations than unknowns, the system of equations is incapable of being either regular or irregular, and the worst-case behavior for F_5 is captured instead by *semi-regular* systems. The degree of (semi-)regularity is now defined as the degree of the first non-positive term in the power series

expansion of $\mathrm{HS}(s) = \frac{\prod_{i=1}^{m}(1-z^{\deg(f_i)})}{(1-z)^n}$, where m is the number of equations and n the number of variables. Note that when $m \leq n$ this formal power series is a polynomial and the Macaulay bound indicates one more than its degree; this is what justifies re-using the term *degree of regularity*. However, while F_5 must reach this degree before it terminates, the degree of the resulting Gröbner basis is typically much smaller.

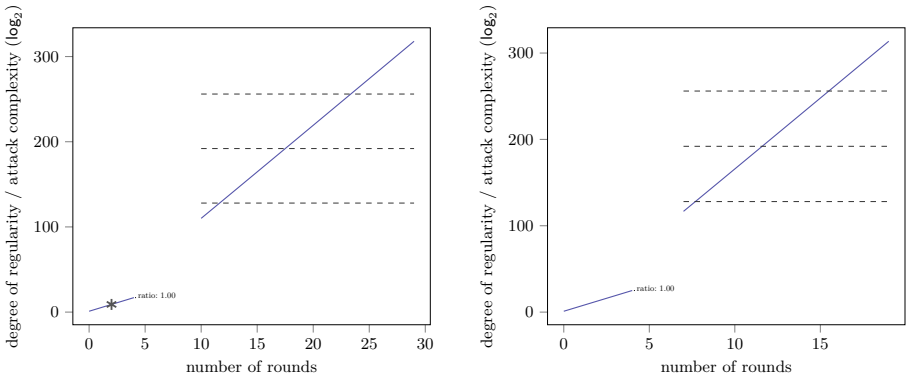
Regardless of whether the system is regular, knowledge of the degree of regularity provides a lower bound on the complexity of computing a Gröbner basis, namely that of running Gaussian elimination on a Macaulay matrix of degree d_{reg} polynomials in n variables. At this point there are $\binom{n+d_{\mathrm{reg}}}{d_{\mathrm{reg}}}$ binomials of degree d_{reg} or less, and $\binom{n+d_{\mathrm{reg}}}{d_{\mathrm{reg}}}^{\omega}$ therefore bounds the attack complexity, where $\omega \geq 2$ is the linear algebra constant — $\omega = 3$ for standard Gaussian elimination; $\omega \approx 2.37$ if fast multiplication techniques [41] are used; and $\omega = 2$ when sparse linear algebra techniques such as Wiedemann's algorithm [40] can be used.

Buchmann *et al.*, writing before the above-mentioned results on the degree of regularity were established, observe that for specially chosen monomial orders, the Gröbner basis comes for free as a result of clever polynomial modeling [16]. The bottleneck of the attack then consists of the monomial order conversion using either FGLM or Gröbner Walk.

In stark contrast, the security rationale underlying our cipher designs is explicit about the designed intractability of the first Gröbner basis computation step. Whatever steps come after might be of greater or lesser complexity and are either way irrelevant to the security consideration. In particular, the security of our ciphers is determined with respect to the Gröbner basis calculation in degrevlex order with $\omega = 2$. The degree of regularity is experimentally tested against that of regular systems of the same dimension for small round numbers. In the case of Vision we observe that the experimental degree of regularity and the degree of regularity of regular systems are equal; in the case of Rescue, we observe a linear relation and extrapolate from there.

B 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 4: 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.

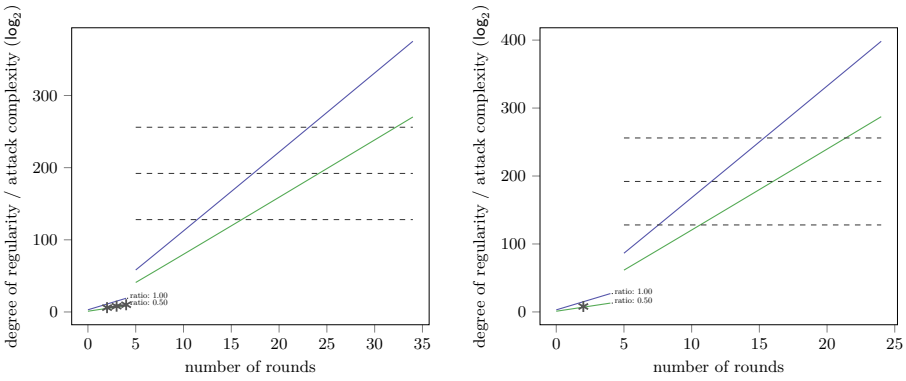
C STARK Intuition

We start be recalling the relevant definitions from [10], along with a simple motivating example.

Scalable Interactive Oracle Proofs (IOPs) and Transparent Arguments of Knowledge (STARKs) like [9,10] 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 $x_i = x_{i-1} + x_{i-2} \mod p$."



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

Figure 5: 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.

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.

D 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 $\mathtt{w}$ *and* length $\mathtt{t}$ *over a field* $\mathbb{F}$ *is an array with* $\mathtt{t}$ *rows and* $\mathtt{w}$ *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* $\mathtt{t} \cdot \mathtt{w}$*.*

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* $\mathtt{d}$*, width* $\mathtt{w}$ *and length* $\mathtt{t}$ *over the field* $\mathbb{F}$ *is a set of multivariate polynomials of total degree at most* $\mathtt{d}$*, with coefficients in* $\mathbb{F}$ *and variable set* $R_{ij}, i \leq \mathtt{w}, j \leq \mathtt{t}$*.*

We point out that the definition of AIR in [10] is slightly more complicated (dealing with boundary constraints and neighborhood sets) but for the purpose of the current work the simpler definition above suffices.

E Algorithms for Masked MPC

We provide here `C++`-like algorithms for the various masking techniques used in Section 7.3.

E.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;
}
```

E.2 Inverse of Sparse Linearized Polynomial

We discuss a technique to efficiently evaluate the inverse of sparse linearized polynomials thanks to the following observations. 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 pseudo-code below shows this procedure more formally.

```
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)
}
```

E.3 α-power and Inverse-α-power

We discuss techniques to efficiently evaluate α-power maps and their functional inverses. Both techniques are explained in a similar manner, we only discuss the technique to evaluate the inverse-α-power map. The participants start by

generating a shared secret mask $[r]$. They then 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 below.

```
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;
}
```

F MPC with Square-and-Multiply

In Section 7.3 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 [24]. 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 a *MiMC-q/q* 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-q/q* with $q = 2^{129} - 45$; 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-q/q* over MPC using square-and-multiply.

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

G Cryptanalytic Strength of *Vision*

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 [35] we find 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 diffusion layer is an MDS matrix applied on m state elements, we have at least $m+1$ active S-Boxes every two steps. When requiring that $n/m \geq 4$ we find that 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.

Algebraic Degree The algebraic degree of a function f is defined as the degree of the largest monomial in the algebraic normal form of f. Ciphers which achieve a low algebraic degree are potentially vulnerable against higher-order differential attacks as introduced by Knudsen [30]. For our construction, the S-Box has algebraic degree $n/m - 1$ after two steps (taking into account both $B(x)$ and $B^{-1}(x)$). The maximal algebraic degree that can be reached by a polynomial in $\mathbb{F}_{2^{n/m}}$ is $n/m - 1$, thus this is achieved already in one round as per our design strategy following [35].

Interpolation Attacks Jakobsen and Knudsen introduced in [29] 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. These attacks lend themselves to meet-in-the-middle variants where the attacker tries to find a concise rational expression in the plaintext or ciphertext. Since $B^{-1}(x)$ is of full degree and dense, we expect that two rounds of *Vision* are sufficient to create a complex rational expression between the plaintext and the ciphertext. To thwart potential meet-in-the-middle attacks we consider three rounds of the cipher to be sufficient in order to resist interpolation 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 a subfield. 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 To deploy an attack on *Vision hash*, one compiles a list of polynomial equations associated with one sponge permutation absorbing one unknown data block and squeezing out one known block. We focus on this case as it captures pertinent use cases.

The following system encodes one full round of *Vision*. Here S_{2i-1} is the intermediate state in the middle of Fig. 1, and K_{2i} and K_{2i-1} represent known constants coming from the key injections. Furthermore, when isolated, $[m]$ denotes the the set $\{1, \ldots, m\}$; but when it is suffixed to a vector or matrix $[i]$ or $[i, j]$ takes the indicated element, and in particular $M^{-1}[i, j]$ takes the (i, j)th element of M^{-1}.

$$\left\{ \begin{array}{ll} S_{2i-2}[j] \cdot B\left(\sum_{k=1}^{m} M^{-1}[j,k] S_{2i-1}[k] - K_{2i-1}[k]\right) - 1 = 0 & j \in [m] \\ S_{2i-1}[j]^4 \cdot B\left(S_{2i-1}[j]^{-1}\right) - S_{2i-1}^4 \cdot \sum_{k=1}^{m} M^{-1}[j,k](S_{2i}[k] - K_{2i}[k]) = 0 & j \in [m] \end{array} \right\}$$

Note that left hand side of the second line is a polynomial in $S_{2i-1}[j]$ as the negative powers are cancelled by the factor $S_{2i-1}[j]^4$ 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 $2mN$ variables and as many equations, with N the number of rounds and all polynomials are of degree 5. Assuming the system is regular we find via the Macaulay bound that $d_{\text{reg}} = 1 + \sum_{i=1}^{2mN}(\mathsf{deg}(f_i) - 1) = 8mN + 1$. Our experiments confirm this degree of regularity for small round numbers.[10]

H Cryptanalytic Strength of *Rescue*

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 the α power map is $(\alpha - 1)$-uniform, the map has a difference propagation probability of at most $\delta = 2^{-\log_2(p)+\log_2(\alpha-1)}$ where the differences are taken over $\mathbb{F}_p$. Since this map is a permutation, its inverse has the same property. 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. Following the stochastic equivalence assumption with the condition that $\log_2(p) \geq 2\log_2(\alpha -$

[10] For more details on our experiments we refer to Appendix B.

1) , we find that for four rounds, or eight steps, of *Rescue*, a differential trail has a maximal differential probability

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

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.

When α is kept small such that $n \leq 2(m+1)(\log_2(p) - \log_2(\alpha))$ with n the bit-level security, the previously discussed security analysis still holds. When α increases, the difference propagation probability increases for x^α and $x^{\frac{1}{\alpha}}$ as now $\delta = 2^{-\log_2(p)+\log_2(\alpha-1)}$. Thus by increasing α , the number of rounds might need to be adapted in order to attain similar security levels.

For a discussion about linear cryptanalysis in prime fields see 4.5.

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 map, the polynomial expression is dense. From [29], 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.[11] Like before we use $[m]$ to denote $\{1, \ldots, m\}$ unless the brackets are a suffix to a vector or matrix, in which case the indicated element is meant.

$$\left\{ \begin{array}{ll} \left(\sum_{k=1}^{m} M^{-1}[j,k](S_{2i-1}[k] - K_{2i-1}[k])\right)^\alpha - P[j] - K_0[j] & i = 1, j \in [m] \\ \left(\sum_{k=1}^{m} M[j,k]S_{2i-1}[k]^\alpha\right) + K_{2i}[j] - \left(\sum_{k=1}^{m} M^{-1}[j,k](S_{2i+1}[k] - K_{2i+1}[k])\right)^\alpha = 0 & \\ & i \in [N-1], j \in [m] \\ \left(\sum_{k=1}^{m} M[j,k]S_{2i-1}[k]^\alpha\right) + K_{2N}[j] - S_{2N} = 0 & i = N, j \in [m] \end{array} \right\}$$

This encoding introduces m new variables and as many new equations of degree α per extra full round. The first step introduces one variable and m equations, whereas the last step introduces no variables and one equation. So along with m variables representing a single state to start from, we have in total $1 + mN$

[11] We explain in more detail how folding is done for *Rescue* in Section 7.1.

variables and $1 + mN$ equations. If the system of equations were regular we would find via the Macaulay bound $d_{\text{reg}} = 1 + \sum_{i=1}^{1+mN}(\mathsf{deg}(f_i) - 1) = (\alpha - 1)(mN + 1) + 1$. Experimentally, we observe the concrete degree of regularity $d_{\text{con}} = \lceil \frac{d_{\text{reg}}}{2} \rceil$ for small round numbers.[12] We extrapolate from here, assuming a constant concrete-to-regular ratio of 2.

[12] The experiments can be viewed in Appendix B.