IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 3, pp. 1–45. DOI:10.13154/tosc.v2020.i3.1-45

Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols

Abdelrahaman Aly[1], Tomer Ashur[1,2], Eli Ben-Sasson[3], Siemen Dhooghe[1] and Alan Szepieniec[1,4]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] TU Eindhoven, Eindhoven, the Netherlands
t.[lastname]@tue.nl
[3] StarkWare Industries Ltd, Netanya, Israel
firstname@starkware.co
[4] Nervos Foundation, Panama City, Panama
firstname@nervos.org

Abstract. While traditional symmetric algorithms like AES and SHA-3 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 algorithms 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 oftentimes different from the considerations arising in the design of software- and hardware-oriented ciphers.

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 the *Marvellous* design strategy which provides a generic way to easily instantiate secure and efficient algorithms for this emerging domain. We then show two examples for families following this approach. These families — *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), and Multi-Party Computation (MPC). These benchmarks show that our algorithms 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

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 create an encryption scheme; or in a PGV construction or a truncated permutation to generate compression functions which in turn can be used to construct hash functions. This last example, hash functions, are a fundamental primitive in their own right for their fitness to approximate a random

Received: 2019-09-01 Revised: 2020-06-01 Accepted: 2020-08-01 Published: 2020-09-28

oracle, and thereby admit a security proof based on this idealization in the random oracle model.

While the security of standard block ciphers and hash functions such as AES, 3DES, SHA2-256, or SHA-3/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 domain 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* [ARS+15], *MiMC* [AGR+16], *Jarvis* [AD18]. The distinguishing feature of these algorithms 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-linear arithmetic operations. These algorithms can thus be characterized as *arithmetization-oriented*, as opposed to *hardware-* or *software-oriented* algorithms which do not have the alternative optimization target.

Arithmetization-oriented cipher design is different from 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 sometimes need to 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 application — but jointly stand in contrast to traditional efficiency metrics such as the ones mentioned above.

To illustrate this difference, consider the performance of the STARK prover when proving the correct evaluation of a hash function. The bottleneck is interpolation and evaluation of a large polynomial, and its degree is the key indicator for the complexity of this task. The STARK paper determines this degree for SHA2 as 41,382 [BBHR18, Tab. 5]; the matching numbers for our *Marvellous* designs (and even for other arithmetization-oriented designs that are not members of the *Marvellous* universe) are about 100 times more efficient (see Table 2) before even taking into account the higher throughput they offer.[1] The one hundred-fold performance improvement justifies transitioning to an arithmetization-oriented hash function — provided that the new hash function is as secure as the old one.

Considering recent trends, arithmetization-oriented cipher design is in its nascency. Rather than blindly optimize for a single vaguely defined metric and shipping the resulting algorithm as soon as possible, it is worthwhile and timely to stop and re-evaluate formerly optimal strategies with respect to this new design space. 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 systematic exploration and mapping of the problem and design landscape that these ciphers inhabit. This exploration gives rise to design strategies, such as the *Marvellous* design strategy which we present here or the Hades design strategy by Grassi *et al.* [GLR+19].

Our ciphers, *Vision*[2] and *Rescue*[3], merely represent a *Marvellous* vantage on our journey. Some additional contributions include a novel approach to ensure resistance to Gröbner basis attacks which is based on the difficulty of finding any Gröbner basis. Whereas earlier works argued their resistance based on the difficulty of computing a term order conversion, our approach has now become the standard in more recent works (see, *e.g.*, [GKK+19]).

This paper is structured in accordance with a progressive refinement of focus. First, in Section 2, we characterize the common features of the advanced cryptographic protocols that arithmetization-oriented ciphers cater to and identify and clarify the exact and various

[1] For a recent comparison in an MPC setting, see [BST20].
[2] In the Marvel comics, Vision is a binary field powered android created by Tony Stark.
[3] In the Marvel comics, Rescue is a prime character and the latest armored avenger.

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 where we introduce the *Marvellous* design approach; 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 give specifications of two designs following the *Marvellous* design strategy and are thus *Marvellous* designs: *Vision* and *Rescue*.

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 other arithmetization-oriented cipher designs with respect to the relevant metrics in these applications.

The paper includes multiple appendices: Appendices A–B provide brief backgrounds on Gröbner basis attacks and Scalable Transparent ARguments of Knowledge (STARKs). These are intended for the uninformed reader and in the sequel we assume familiarity with these concepts. In Appendix C we explain the masking techniques we considered when benchmarking our algorithms in an MPC setting. Users implementing our algorithms in such a setting can consult this appendix. Finally, Appendices D–E provide instances for *Vision* and *Rescue*. These are offered as targets for cryptanalysis as well as for readers interested only in "the bottom line". We also provide an "instance generator" allowing users to create new instances from a given parameter set. This can be found in [SDS19].

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 characterization 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 [Raz87] as a mechanical method in the context of computational complexity and first applied to cryptographic protocols by Lund *et al.* [LFKN90]. 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 if a non-deterministic 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 *non-procedural computation*, in which the state of the system at the next computational step cannot be described as having been solely 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 non-procedural computations.

2.1 Zero-Knowledge Proofs

A zero-knowledge (ZK) proof system is a protocol between a prover and a verifier whereby the former convinces the latter that their common input ℓ is a member of a language $\mathcal{L} \subset \{0,1\}^*$. The proof system is complete and sound with soundness error ϵ if it guarantees that the verifier accepts (outputs 1) when $\ell \in \mathcal{L}$ and rejects with probability $\geq 1 - \epsilon$ when $\ell \notin \mathcal{L}$. When this soundness guarantee holds only against computationally bounded provers we call it an *argument* system. The proof system is zero-knowledge if the transcript is independent of the membership or non-membership relation.[4] We are concerned here with languages $\mathcal{L}$ that capture generic computations in different models of computation.

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

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

Following [BBHR18], 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 $\mathsf{poly}(n) + \mathsf{poly}\log T(n)$.
- *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 [BBHR18], following theoretical constructions [BCGV16, BCF+16] (cf. [BBC+17] 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 [BBHR18]. 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 B.[5]

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

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

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 $\vec{X} = (X_1, \ldots, X_{\mathsf{w}})$, $\vec{X'} = (X'_1, \ldots, X'_{\mathsf{w}})$, representing, respectively, the current and next state of the computation, such that a transition from state $\vec{s} = (s_1, \ldots, s_{\mathsf{w}}) \in \mathbb{F}_q^{\mathsf{w}}$ to state $\vec{s'} = (s'_1, \ldots, s'_{\mathsf{w}}) \in \mathbb{F}_q^{\mathsf{w}}$ is valid iff all polynomials in $\mathcal{P}$ evaluate to 0 when the values $\vec{s}, \vec{s'}$ are assigned to the variables $\vec{X}, \vec{X'}$, respectively. (See Appendix B for an example.)

To maximize the efficiency of ZK-STARKs, we wish to minimize the three main parameters of the AIR: the computation time t, the state width w and the maximal degree d of an AIR constraint (polynomial) in $\mathcal{P}$. While 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.*, [PGHR13, BCG+13, WSR+15]. Such circuits are specified by directed acyclic graphs with labeled nodes and edges. The edges, or *wires*, have a value taken from some ring; the nodes, or *gates*, perform some operation from that ring on the values contained by its input wires and assign the corresponding output value to its output wires. An assignment to the wires is valid if and only if for every gate, the value on the output wires matches that gate's operation and the values on its input wires. In the context of zero-knowledge proofs, the prover generally proves *knowledge* of an assignment to the input wires of a circuit computing a one-way function, meaning that the corresponding output matches a given public output. Alternatively, the prover can prove *satisfiability* — that there exists a corresponding input — which makes sense in the context where it is also possible for no such input to exist.

Recent years have seen a growing focus concerning Quadratic Arithmetic Programs (QAPs) [GGPR13] and Rank-One Constraint Satisfaction (R1CS) systems [BCG+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 [GGPR13]. Additional transparent systems such as Ligero [AHIV17], Bulletproofs [BBB+18] and Aurora [BCR+19] 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.

2.2 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 several 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 multiplicative count, and which can be realized with low offline and online complexities, are masking operations such as the technique suggested by Damgård *et al.* [DFK$^+$06]. This 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 it multiplies $[a]$ with $[r]$, opens $[ar]$, and then locally raises this masked plaintext value 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 [BB89].

We extend this range of techniques in two ways. First, we adapt the technique of Damgård *et al.* for exponents of the form $1/\alpha$, with α small; while the online complexity is the same, our technique reduces the offline complexity by exploiting the small size 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 in Section 7. 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) [oS77]. Since then, methods have been developed for designing new block ciphers. A number of families for the basic operations (*e.g.*, ARX) were explored as well as general purpose structures (*e.g.*, (G)Feistel, or SPN). The cryptographic properties of commonly used components were analyzed (*e.g.*, S-boxes with high non-linearity, a linear layer with fast diffusion) and methods for determining a safe number of rounds based on theoretical arguments (*e.g.*, the wide trail strategy) or using automatic tools (*e.g.*, MILP and SAT-solvers [MWGP11, MP13, KLT15]) were developed. These methods, if used properly, reduce the effort in the design cycle and result in efficient algorithms that are resistant to known attacks.

In contrast, arithmetization-oriented ciphers necessitate different design considerations. In this section, we focus on considerations that are of special interest when designing arithmetization-oriented algorithms. This is not to say that one can never find these considerations in traditional cipher design, only that they usually take a less prominent role. This section is independent of our cipher designs. To the extent that it raises questions or concerns, we address these in the context of the *Marvellous* design strategy in Section 4.

3.1 Non-Procedural 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, these phenomena seem to undermine the constraining character of procedural evolutions or violate it altogether. We call these phenomena *non-procedural computations.*

It is possible to design and define ciphers in terms of non-procedural 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 non-procedural operations. As a result of this design decision, 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 power map $x \mapsto x^{q-2}$ (which is often used to compute the multiplicative inverse function), for some $x \in \mathbb{F}_q$. When the field is large, so is the exponent, and as a result a procedural evaluation (*e.g.*, by calculating a GCD or via square-and-multiply) is expensive. We show how this operation is captured efficiently by non-procedural computation in various arithmetic modalities.

In the case of zero-knowledge proofs, the particular variant of non-procedural 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 non-procedural 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 for each inversion a shared value, $[a]$ satisfying $a \neq 0$. Then in the online phase, $[ax]$ is opened and the result is inverted locally before being multiplied with $[a]$, yielding $[y] = (ax)^{q-2}\,[a]$. This description ignores the special care necessary when $x = 0$ but the point remains that the number of online secret shared multiplications (which are expensive) is independent of the power to which the shared secrets are raised.

Another example of non-procedural 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 procedurally. 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 non-procedural computation adds another dimension to cipher design by allowing operations that would have been expensive if implemented directly in software or hardware.[6]

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*

[6]This point is actually a little more subtle. While this operation still need to run on "traditional" hardware, the modalities of computations we are concerned with actually abstract this part and only count the number of field operations. This is, for better or worse, a common practice in this area of research and we follow suit.

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 illustrate 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 the 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 a non-procedural 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 $\log_d(\alpha)$ AIR constraints.

Importantly, and unlike in the case of traditional cipher design, the size of the field over which the cipher is defined is virtually 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. Conversely, in traditional cipher design and the cost metrics normally associated with it, such a change would increase the circuit size, RAM, latency, and throughput.

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 their bit expansion, which is unnecessarily expensive in our target settings. This observation rules out entire classes of designs, *e.g.*, ARX, or bit-oriented algorithms and arithmetization-oriented ciphers must sacrifice the security benefits conferred by mixing algebras to achieve even the most basic level of efficiency.

3.3 Cryptanalytic Focus

In traditional cipher design, statistical attacks — particularly, differential and linear cryptanalysis — are considered two of the strongest tools in the cryptanalytic toolbox. Although other types of attacks have been shown in some cases to deliver interesting results, they systematically receive less attention in the literature.[7] However, the opposite seems to be the case for arithmetization-oriented ciphers. The flexibility in choosing the field size, the gratuitous nature of scalar multiplication, and non-procedural computation allows killing statistical attacks in a rather small number of rounds rendering standard security arguments such as the wide trail strategy less important.

The optimization of ciphers for arithmetic modalities of computation does have 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 non-procedural computation). While it is true that any function from finite fields to finite

[7]Recent times have seen some change to this trend with the rise of other types of attacks; still, statistical attacks seem to be receiving more focus than others by a large margin.

fields can be represented by a polynomial, the problem is that arithmetization-oriented ciphers attempt, as a design goal, to 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 [JK97], higher-order differentials [Knu94, Lai94], and the GCD attack [AGR$^+$16, §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 the polynomial is 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 quantify in terms of complexity. This difficulty stems from a variety of sources:

- 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 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 many parasitical solutions in the field closure despite having dimension zero. 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. Since the variety is zero-dimensional, there must be at least one univariate basis polynomial at this point; 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 [FGLM93]; however an alternative algorithm called Gröbner Walk does not have a rigorous complexity analysis and yet is observed to outperform FGLM sometimes in practice [BPW06a].

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

[8] Interestingly, AES, which is surprisingly arithmetizible considering that it was not designed as such, also admits certain algebraic attacks [CMR06].

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 seems to be *the* crucial concern raised by arithmetization-oriented ciphers, and no such proposal is complete without explicitly addressing it. This is yet another difference from traditional designs where Gröbner basis attacks are mostly irrelevant and it is the statistical attacks that require special care by the designer.

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. By linking the efficiency and security to the same cost metric this correspondence hints towards a possible limitation on the attainable efficiency for a fixed security level (and vice versa).

The relative importance of Gröbner basis attacks is illustrated by *Jarvis* [AD18] and *MiMC* [AGR+16], 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* [ACG+19]. An investigation of *MiMC* using the same attack was argued to be infeasible [ACG+19, Sec. 6].[9] 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, as a countermeasure against Gröbner basis attacks, relying on the large number of parasitical solutions, or on the according complexity of term order change, is a new security argument as well as 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 4.2.3 we answer this question positively by developing a novel framework for providing such an argument.

3.4 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. It is possible that we overlooked other advanced cryptographic protocols employing arithmetic modalities of computation, or that some 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.

[9] We note that a recently published work [EGL+20] showed that the algebraic degree of this cipher grows slower than originally believed which may have implication still for a Gröbner basis attack against the cipher.

Lastly, we note that the field of algebraic attacks against symmetric-key algorithms appears to be underexplored for the most part. As a result, it is difficult to make a compelling security argument valid for the entire class 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 (Sections 4.2.2–4.2.3).

4 The *Marvellous* Design Strategy

Following the discussion in Section 3 we outline a framework for designing secure algorithms which are efficient in arithmetization-oriented applications; this is the *Marvellous* design strategy. In this section we explain and motivate the decisions made in relation to this strategy and defer the realization of specific families to Sections 5–6 with instances for some real world use cases provided in Appendices D–E

4.1 General Structure and Design Approach

In essence, a *Marvellous* design is a substitution-permutation (SP) network parameterized by the tuple

$$(q, m, \pi, M, v, s).$$

The state is an element in the vector space $\mathbb{F}_q^m$, with q either a power of 2 or a prime number, $\pi = (\pi_0, \pi_1)$ the S-boxes, M an MDS matrix, v the first step constant, and s the desired security level. For security reasons we require $m > 1$ and $\log_2(q) > 4$. $\pi = (\pi_0, \pi_1)$ should be selected in such a way that π_0 (resp., π_1) has a high degree when evaluated in the direction of the encryption (resp., decryption) function. All these constraints will be motivated in the sequel. Finally, to ensure the existence of the MDS matrix M we also require that $2m \leq q$.

Owing to the recent proliferation of arithmetization-oriented protocols in the real world (*e.g.*, [Staa, Ben, BST20]; for a more elaborate survey see [ARS+15, Sec. 2]), the design of suitable cryptographic algorithm is of more than just academic interest. We begin by outlining our design principles leading to the *Marvellous* design strategy, listed by order of importance:

1. Security is more important than efficiency. This design principle influenced several of our design choices including the selection of structure, the use of a "heavy" key schedule, and the addition of generous safety margins.

2. Simplicity and robustness. Given that the suggested designs are expected to be used in the real-world and implemented by users coming from various backgrounds, we tried to minimize the design strategy to a small set of simple choices. We argue that, if followed, any compliant choice results in a secure algorithm. Of particular importance in our opinion is the robustness against "wrong" decisions by the user. We made it a priority to ensure that the design is "fool-proof", *i.e.*, that it is hard for a user to make an accidental parameter choice that would adversely affect the security of the resulting instance. In other words, the designs come with no fine prints nor are there any delicate parameter choices. Incidentally, this gives users more flexibility to construct instances most suitable for their use cases.

3. Justifiable. Understanding that the domain of arithmetization-oriented ciphers is rather new and mostly unexplored we aim to motivate in simple language all design decisions and security arguments. This facilitates an easy start for third party cryptanalysts who are interested in evaluating the security of our algorithms, as well as for advanced users who may choose to deviate from parts of the design strategy if they feel confident to do so. Furthermore, in the event of a newly discovered attack,

it is easy to pinpoint the failed principle and adapt it rather than reiterate the entire design strategy.

4. Efficiency. Having the previous three design principles satisfied, we can finally focus our attention on ensuring that the resulting algorithms are also efficient in their respective use cases. In other words, at this point we sieved through possible unconditionally secure, sufficiently simple and robust, and properly justified design approaches in search for the most efficient one with respect to our cost metric. We believe that our prudent and conservative choices at this early stage will prove useful against third party cryptanalysis over time.

4.1.1 Primitives and constructions

While most of the applications of arithmetization-oriented ciphers are in the form of hash functions, we prefer to present a more generic primitive. In particular, the starting point is a family of permutations where the family defines a base permutation, and a member of the family is parameterized by a key. We specify a key schedule function which determines the round keys, together with the base permutation, these used to form an iterated Even-Mansour block cipher.

The key schedule is also used to generate the round constants. In particular, setting the key to zero (see Section 4.1.7) allows to use the resulting permutation as the underlying primitive for a sponge or duplex function. Fixing the output length results in a hash function.

In Section 4.2 we show how to argue the security of a *Marvellous* permutation generically. Later on, we specify this analysis with respect to *Vision* and *Rescue* and focus on concrete Gröbner basis attacks in the hash setting. We claim that the block cipher setting offers a similar or higher security level, because due to the key schedule, the system of polynomial equations grows by about a factor two. *Marvellous* permutations can be used to realize additional functionalities even beyond block ciphers and hash functions, such as MACs, stream ciphers, and PRNGs, but similar security arguments for these constructions would require additional analysis and are out of scope for this paper.

4.1.2 The state

The state of a *Marvellous* design is viewed as a vector of $m > 1$ field elements $x_0, \dots, x_{m-1}$, *i.e.*, a *Shark*-like structure [RDP+96]. Since the main cost metrics are minimizing the multiplicative complexity and keeping the multiplicative depth low, we felt that the design would benefit from the fast diffusion offered by this structure compared to *e.g.*, Square or (G)FN structures. The fast diffusion of a *Shark*-like structure would normally increase the circuit cost. However, as our cost metrics are not influenced by the size of the matrix or the number of scalar multiplications, the efficiency of the design does not suffer by this choice.

In the course of our work, we observed unexpected and suspicious behavior of some algebraic properties when setting $m = 1$. Investigating this further revealed a qualitative difference between the cases of $m = 1$ and $m > 1$. In lieu of a good explanation to this behavior and in accordance with the design rationale we decided to restrict the designs to $m > 1$. Our findings were later confirmed by third party analysis for some $m = 1$ designs [ACG+19, EGL+20].

4.1.3 Round function

Marvellous designs are formed by N iterations of a round function which takes the previous state and a subkey as inputs and outputs a new state. The inputs to the first round are the plaintext and the master key, and the output of the last round is the ciphertext.

Subsequent subkeys are derived from a master key by means of a key schedule as explained in Section 4.1.7.

A single round consists of 2 steps. Each step employs three layers: S-box, linear, and subkey injection. The S-box layer (Section 4.1.4) is full, *i.e.*, it applies an S-box π_i to each of the m state elements. The linear layer (Section 4.1.5) is a multiplication between a matrix M and the output of the S-box layer. The key injection layer adds the output of the corresponding step in the key schedule algorithm (Sections 4.1.7) into the state. Note that even for keyless algorithms, the output of the key schedule is non-empty thus produces step constants (Section 4.1.6). All algebraic operations are realized using the field's native operations.

4.1.4 S-boxes

In each round of a *Marvellous* design a pair of S-boxes (π_0, π_1) is used with π_0 used in the S-box layer of even steps (starting from 1) and π_1 in the S-box layer of odd steps. Each S-box is a simple power map x^α possibly composed with an affine transformation.

The difference between π_0 and π_1 is in their degree. They should be chosen such that π_0 has a high degree when evaluated forward (*i.e.*, in the direction of the encryption) and a low degree when evaluated backward (*i.e.*, in the direction of the decryption). The other S-box, namely π_1, is chosen with the opposite goal (*i.e.*, to have a low degree in the forward direction and a high degree in the backward direction). This choice serves to achieve three goals: (i) no matter which direction an adversary is trying to attack, the degree is guaranteed to be high; (ii) it results in the same cost for the encryption and decryption functions, and (iii) owing to non-procedural computation, the low-degree representation of each S-box can be evaluated efficiently.

The choice of power map S-boxes is motivated by two reasons. First, owing to the seminal work of Nyberg [Nyb93] the cryptanalytic properties of power maps are well understood and allow to make solid security arguments which have also proved themselves in practice [DR02].

For algorithms intended to work over binary fields the power map is composed with an affine transformation. For efficiency reasons, we recommend to use an $\mathbb{F}_2$-affine linearized polynomial, *i.e.*, a polynomial of the form

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

This affine transformation B is drawn randomly from the set of all $\mathbb{F}_2$-affine linearized polynomials with a fixed degree in $\mathbb{F}_{2^n}[x]$. Such a polynomial is a permutation over $\mathbb{F}_{2^n}$ if and only if its linear part only has the root 0 in $\mathbb{F}_{2^n}$. Specifically, we suggest to generate the coefficients by employing SHAKE-256 to expand a short seed into a long sequence from which the coefficients can be taken (with rejection as necessary to ensure that the polynomial is invertible). We recommend to use $\mathbb{F}_2$-affine polynomials of degree 4 but other degrees are also allowed as long as B^{-1} is verified to be of maximal degree or very close to it.

In the two families we present in this paper, π_0 is obtained directly from π_1. In the case of *Rescue* we have $\pi_1 = x^\alpha$ and $\pi_0 = x^{1/\alpha}$. In *Vision* the S-boxes take the form $\pi_1 = B(x^{-1})$ and $\pi_0 = B^{-1}(x^{-1})$. We stress that while we consider such choices a good practice which positively affects the efficiency, this does not have to be the case and the pair (π_0, π_1) can be chosen in other ways as long as they jointly ensure that the degree of a single round is sufficiently high in both directions.

4.1.5 Linear layer

As in other designs, the purpose of the linear layer is to "spread" locally good properties onto the entire state. Having chosen the SPN approach, and being unable to manipulate individual bits due to the arithmetization-oriented nature of the designs, a matrix multiplication is the natural choice to serve as a linear layer.

A special class of matrices often used in SPN's are MDS matrices. In our setting, an MDS matrix ensures a maximal branch number thus enabling full diffusion in a single round. The propagation of such matrices is well-studied for both statistical and algebraic properties (*e.g.*, polynomial degree). Thus, MDS matrices seem *Marvellous* enough to be used in our designs.

To ensure that the MDS matrix exists, we require that the native field $\mathbb{F}_q$ on which it operates is at least 5-bit wide and that $2m \leq |\mathbb{F}_q|$ (*i.e.*, $\log_2(q) > 4$ and $2m \leq q$). Following the design rationale, none of the security arguments is sensitive to the choice of a specific MDS matrix. Since scalar multiplication bears no cost in arithmetization-oriented ciphers, efficiency also does not play a role here. We conclude that any MDS matrix can be used to realize the linear layer of a *Marvellous* design as long as it has the right dimensions.

Stressing once more that all MDS matrices are *Marvellous*, we offer to use $m \times 2m$ Vandermonde matrices using powers of an $\mathbb{F}_q$ primitive element. This matrix is then echelon reduced after which the $m \times m$ identity matrix is removed and the MDS matrix is obtained. This procedure was used to generate the linear layer of all but one of the instances given in Appendices D–E.

4.1.6 Round constants

When designing a new symmetric-key algorithm, constants are injected into the state to thwart certain attacks (*e.g.*, rotational cryptanalysis, invariant subspace attacks, etc.) by breaking possible symmetries and/or similarities between parts of the algorithm.

The only thing we require from our constants is that they do not belong to any subfield of $\mathbb{F}_q$, nor be rotational-invariant. Concretely, we propose the following method for generating them: we use SHAKE-256 to expand a short seed into a long sequence from which one samples the first constant (with rejection as necessary to ensure that the value is a member of $\mathbb{F}_q$ and not a member of any subfield of $\mathbb{F}_q$). 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 [SDS19].

In our case, we opt to adding state-wide constants to the key schedule, resulting in a "fresh" value injected into the state in every round. Note that even for unkeyed constructions, (e.g., 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.

4.1.7 Key schedule

The key schedule reuses the round function. The master key is fed through the plaintext interface and the round constants (Section 4.1.6) are added where the subkey would normally be injected. The subkeys are then determined as the value of the state immediately following the injection of the constant.

In recent years, driven by the advent of lightweight cryptography, complex key schedules have fallen out of favor. Following the *Marvellous* design rationale, we decided to take the opposite approach here, namely using a heavy key schedule (*i.e.*, with cost of the same order as the algorithm itself). This complexity is motivated by the following reasoning:

- 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 for arithmetization-oriented ciphers is hashing and in this case it is possible (*e.g.*, by building a sponge function) to completely hide the overhead of the key schedule as its input is a fixed key. In other cases it may be possible to amortize the cost of the key schedule over the cost of the entire execution or offset it to an offline phase.

- 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.[10]

- A less straightforward Gröbner basis attack on the block cipher targets the injected subkeys rather than the master key. However, as these injected subkeys are different, they must be treated as independent variables. Consequently, the number of plaintext-ciphertext pairs that are necessary to uniquely determine these subkeys must be equal to the number of 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.

When using a *Marvellous* design as an unkeyed primitive (*e.g.*, as a permutation in a sponge function), the m field elements of the master key are all set to zero and the key schedule is invoked to process the step constants and output fresh values to be used as inputs to the step function.

4.1.8 Number of Rounds

To set the number of rounds for a given parameter set, we consider ℓ_0, the maximal number of rounds that can be generically attacked by any of the attacks in Section 4.2 (for a summary of these attacks see Table 1) and ℓ_1, the instance-specific number of rounds that can be attacked by a Gröbner basis attack.

Our analysis shows that for reasonable parameter choices, statistical attacks, as well as most algebraic attacks, do not extend beyond a handful of rounds. For all parameters we considered, $\ell_0 \ll \ell_1$ and we discuss the instance-specific analysis against Gröbner basis attacks in Sections 5–6. These examples can be consulted by users generating new *Marvellous* families.

Having determined ℓ_0 and ℓ_1, we set the number of rounds to be

$$2 \cdot \max(\ell_0, \ell_1, 5)\,,$$

i.e., we take the number of rounds covered by the longest reaching attack and double it, with a minimum of $2 \cdot 5 = 10$ rounds. We call 5 a *sanity factor* and its purpose is to ensure that aggressive optimization attempts do not result in trivially weak instances.

4.2 Security

We now give an overview of the *Marvellous* security countermeasures applied to defend algorithms following this design strategy against various attacks. In essence, we see that

[10]We also mention in Section 4.2.3 that showing resistance to Gröbner basis attacks in the unkeyed case implies same resistance or better for the keyed case.

for reasonable parameter sets the design strategy provides resistance against statistical and structural attacks, as well as against most algebraic attacks, already after a small number of rounds. We provide below a generic security argument against these attacks with a summary provided in Table 1.

The limiting factor in determining a safe number of rounds for meaningful instances appears to be the resistance to Gröbner basis attacks which we were not able to argue generically. Instead, we developed a novel framework for arguing resistance against said attacks. In Sections 5–6 we employ this framework to determine the resistance of the *Vision* and *Rescue* families, respectively. These analyses can be consulted by users interested in generating their own *Marvellous* families after also consulting Table 1 to ensure that theirs is not the edge case where the Gröbner basis attack is outperformed by one of the other ones.

Note that the security analysis below pertains only to the *Marvellous* algorithm as a permutation proper, *i.e.*, we show that the permutation is indistinguishable from a random one. In practice, this permutation will be used within a mode of operations or a construction which will have their own security claims. We envision that the most common use would be as a primitive to a sponge construction generating a hash function, in which case the generic security of the sponge is given by $\log_2(\sqrt{q})\min(r_q, c_q)$ with r_q the arithmetic rate and c_q the arithmetic capacity; we elaborate on the proper way to employ a *Marvellous* permutation in a sponge construction in Section 4.3. The security of the hash function stems from reducing the security of the sponge construction to that of the primitive (*i.e.*, the underlying permutation) and the argument is completed by observing that this permutation is indistinguishable from random. The interested reader is further referred to the work of Beyne et al. [BCL+20] for a third party cryptanalysis of *Rescue* as a hash function, and to [BCD+20] for a more general analysis including both *Marvellous* families.

Table 1: Resistance to cryptanalytic attacks. Each row in the table denotes an attack class with each cell describing the maximal number of rounds that can be covered by this attack with respect to a security parameter s.

Type of attack	Binary fields (x^{-1})	Prime fields (x^{α})
Differential Cryptanalysis	$\frac{2s}{\log_2(q^{m+1})-2\cdot(m+1)}$	$\frac{2s}{\log_2(q^{m+1})-\log_2((\alpha-1)^{m+1})}$
Linear Cryptanalysis	$\frac{s}{\log_4(q^{m+1})-2\cdot(m+1)}$	-
Higher Order Differentials	$\frac{\log_2(s)}{\log_2(n-1)}$	-
Interpolation	3	3
Gröbner Basis	Sec. 5	Sec. 6

4.2.1 Statistical Attacks

A common security argument for SPN's is the wide trail strategy [DR02]. The argument uses two quantities: an upper bound for the best propagation probability of the statistical property through a single S-box, and a lower bound for the minimal number of active S-boxes. Then, the former is raised to the power of the latter to obtain a (lower) bound on the probability of the best differential characteristic in the cipher.

Building on the work of Nyberg [Nyb93] 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 $\mathbb{F}_q$ is fixed. An interesting observation here is that these quantities improve directly (from the designer's point of view) as a result of increasing the field size. For example, the maximal difference propagation probability of an active S-box in AES (*i.e.*, computing $1/x$ for $x \in \mathbb{F}_{2^8}$) is $\delta_{2^8} = 2^{-8+2} = 2^{-6}$. Comparing this to a hypothetical AES-like cipher where the state consists of elements in $\mathbb{F}_{2^{128}}$ we get $\delta_{2^{128}} = 2^{-128+2} = 2^{-126}$. We see that by merely changing the native field and nothing else, the differential uniformity of the S-box is improved by a factor of 2^{-120}. The situation is similar for linear cryptanalysis.

Bounding the number of active S-boxes: The number of active S-box in a single round follows directly from the dimensions of the MDS matrix being used in the linear layer. For m the number of field elements in the state, at least $m + 1$ S-boxes are active in every two steps (*i.e.*, in one round).

Bounding the transition probabilities: The bound on the best propagation probability depends on the type of field being used (binary vs. prime) and is a function of the power map α.

Binary fields ($q = 2^n$; $\alpha = -1$): In the language of [Nyb93, §4] this is the inversion mapping

$$f : \mathbb{F}_q \to \mathbb{F}_q : x \mapsto x^{q-2} \ ,$$

or in rational form

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

with $\delta = 2^{-\log_2(q)+2}$ and $|\lambda| = 2^{-\lceil \log_2(q)/2 \rceil + 1}$. Since the MDS matrix activates at least $m + 1$ S-boxes in each round we find that the probability of any N-round differential characteristic is at most

$$2^{N(m+1)(-\log_2(q)+2)} \ . \tag{1}$$

Since $\log_2(q) > 4 \Rightarrow (-\log_2(q) + 2) < 0$ and $0 < 2^{N(m+1)(-\log_2(q)+2)} < 1$. Denoting the security parameter by s we seek to bound the probability of the best differential characteristic to be at most 2^{-2s} ($2s$ is used in the exponent in order to account for differential clustering effects). Substituting the security parameter into (1) and solving for N we get

$$\begin{aligned} 2^{N(m+1)(-\log_2(q)+2)} \leq 2^{-2s} \Rightarrow N &\geq \frac{-2s}{(m+1)(-\log_2(q)+2)} \\ \Rightarrow N &\geq \frac{2s}{\log_2(q^{m+1}) - 2 \cdot (m+1)} \ . \end{aligned} \tag{2}$$

Analogously to (1) we see that any N-round linear trail has absolute correlation at most

$$2^{N(m+1)(-\lceil \log_4(q) \rceil + 1)} \leq 2^{N(m+1)(-\log_4(q)+2)} \ . \tag{3}$$

Since a linear attack requires data complexity proportional to the squared inverse of the correlation we set

$$\begin{aligned}(2^{N(m+1)(-\log_4(q)+2)})^2 \leq 2^{-2s} \Rightarrow 2^{N(m+1)(-\log_4(q)+2)} &\leq 2^{-2s/2} \\ \Rightarrow N &\geq \frac{-s}{(m+1)(-\log_4(q)+2)} \\ \Rightarrow N &\geq \frac{s}{\log_4(q^{m-1}) - 2\cdot(m+1)}\end{aligned} \tag{4}$$

Prime fields (q prime; α prime; $\gcd(q-1,\alpha)=1$): In this case we use the power maps

$$f : \mathbb{F}_q \to \mathbb{F}_q : x \mapsto x^\alpha$$

and

$$f^{-1} : \mathbb{F}_q \to \mathbb{F}_q : x \mapsto x^{1/\alpha}$$

which exist and are both permutations if and only if $\gcd(q-1,\alpha)=1$. Again from [Nyb93] we know that in this setting the α power map is $(\alpha-1)$-uniform and has a difference propagation probability at most $\delta = 2^{-\log_2(q)+\log_2(\alpha-1)}$ where the differences are taken over $\mathbb{F}_q$. Having required $\gcd(q-1,\alpha)=1$, the $1/\alpha$ power map is the functional inverse of x^α (*i.e.*, $x = (x^\alpha)^{1/\alpha}$) and is therefore also $(\alpha-1)$-uniform. Considering the $m+1$ active S-boxes per round, we find that the difference propagation probability of any N-round differential characteristic is at most

$$2^{N(m+1)(-\log_2(q)+\log_2(\alpha-1))} \quad . \tag{5}$$

Observing that $\log_2(q) > \log_2(\alpha-1)$ is always true we have $\log_2(\alpha-1)-\log_2(q) < 0 \Rightarrow$ $0 < 2^{N(m+1)(-\log_2(q)+\log_2(\alpha-1))} < 1$. Substituting the security parameter 2^{2s} into (5) and solving for N as before, we get

$$\begin{aligned}2^{N(m+1)(\log_2(\alpha-1)-\log_2(q))} \leq 2^{-2s} \Rightarrow N &\geq \frac{-2s}{(m+1)(\log_2(\alpha-1)-\log_2(q))} \\ \Rightarrow N &\geq \frac{2s}{\log_2(q^{m+1})-\log_2((\alpha-1)^{m+1})} \quad .\end{aligned} \tag{6}$$

Linear cryptanalysis in prime fields is more complicated (see also [BCD+20]). Normally, linear cryptanalysis searches for a linear combination of input-, output-, and key bits that is unbalanced. As such, linear cryptanalysis seems tailored to work over binary fields. No obvious analogue to this behavior exists in prime fields. However, we stress that we do not have a rigorous argument for the inapplicability of linear cryptanalysis in this setting and the questions how to lift linear cryptanalysis to this setting and how many rounds, if any, can be covered by the attack, remain open.

4.2.2 Structural and Algebraic Attacks

Self-similarity attacks. Self-similarity attacks work by splitting an algorithm into multiple sub-algorithms that are similar to one another, for some definition of similarity (hence "self-similarity"). This allows to attack one of the sub-algorithms and use the self-similarity to cleverly link this part with the others. A straightforward way to resist this class of attacks is to inject round constants which break the self-similarity. In Section 4.1.6 we discussed the conditions that step constants must satisfy and suggested a way to generate them.

Invariant Subfield Attacks The invariant subfield attack works if there exist two subfields $\mathbb{F}_{q_1} \subset \mathbb{F}_q$ and $\mathbb{F}_{q_2} \subset \mathbb{F}_q$ such that for any input to the round function $x \in \mathbb{F}_{q_1}$, the corresponding output satisfies $y \in \mathbb{F}_{q_2}$. The two subfields can be the same or different. This invariant subfield attack is only relevant for binary fields since, by definition, when q is prime it has no non-trivial subfields. However, for $q = 2^n$ an adversary might be able to attack the cipher by making it work over one of the subfields. We require that the coefficients of the affine polynomial B used to construct (π_0, π_1) do not lie in any subfield of $\mathbb{F}_{2^n}$ thus frustrating the attack. In addition we require that the step constants are not members of any subfield.

Higher Order Differential Cryptanalysis. In binary fields, the algebraic degree of a function f is defined as the degree of the monomial with the highest degree when f is given in algebraic normal form. Ciphers which achieve a low algebraic degree are potentially vulnerable to higher-order differential attacks [Knu94, Lai94]. The resistance against this attack, for *Marvellous* designs which operate over binary fields, comes from the S-box. Taking into account both $B(x)$ and $B^{-1}(x)$, the algebraic degree is $n - 1$ after a single round and density is assured by repeating the round. Since $n - 1$ is also the maximal algebraic degree that can be reached by a polynomial in $\mathbb{F}_{2^n}[x]$, we expect the algebraic degree of the state after r rounds to be $(n - 1)^r$. For a security parameter s, we require that $s \leq (n - 1)^r$.

Interpolation Attacks. Interpolation attacks [JK97] are yet another class of attacks exploiting the low degree of an algorithm. Here, the attacker tries to reconstruct the polynomial describing the algorithm from input/output pairs by means of Lagrange interpolation. Due to the complexity of calculating GCD's or Lagrange interpolation being linear in the degree of the polynomial, a way to avoid the attack is to ensure that the polynomial representations of the algorithm is dense and of high degree. In a *Marvellous* design, at least one of the power maps in (π_0, π_1) is of high degree thus ensuring a high rational degree of the polynomial expression which, if the round is repeated, makes for a dense polynomial expression. For binary fields, the affine polynomials $B(x)$ and $B^{-1}(x)$ ensure this property also for $\mathbb{F}_2$, *i.e.*, for the base field. Interpolation attacks lend themselves to meet-in-the-middle variants and we conclude that an interpolation attack is frustrated after at most three rounds.

4.2.3 Gröbner basis attacks

Gröbner basis attacks are of particular interest for arithmetization-oriented algorithms as their complexity seems to be coupled with the efficiency of the cipher itself. In other words, since both the efficiency of the cipher and the complexity of the attack are captured by the multiplicative complexity, attempting to improve the former might also make the latter more feasible. While approaches for quantifying and ensuring resistance against other attacks have been heavily studied in the literature, the theory behind Gröbner basis attacks does not appear to be sufficiently advanced to offer similar tools.

Unable to devise a generic security argument, what we provide instead is a novel framework for determining the resistance against Gröbner basis attacks of a given algorithm. Contrary to other works in this domain, our framework does not make a-priori assumptions on either the polynomial system or the number of parasitical solutions. Instead, it extrapolates the polynomial system's properties from empirical data. We offer this framework as a first step towards a systematic approach to resisting Gröbner basis attacks.

For a more elaborate discussion about Gröbner basis attacks, the interested reader is referred to Appendix A. For the present discussion, it suffices to recall the three basic steps involved in deploying such an attack:

(i) computing a 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 principle to derive a safe number of rounds:

resistance to Gröbner basis attacks should come from the infeasible complexity of computing the Gröbner basis in degrevlex order in step (i).

This principle guarantees that the cipher's security against Gröbner attacks 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 since the attacker has to get past step (i) in order to get to step (iii). Interestingly, with this approach, the number of parasitical extension field solutions required for an infeasible univariate factorization is no longer a constraining factor in determining the number of rounds; this appears to result in more efficient algorithms compared to other approaches.

In order to guarantee that finding the first Gröbner basis is prohibitively expensive, we implement the algorithm and a Gröbner basis attack, and observe the degree of regularity experimentally for a small number of rounds. 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, setting $\omega = 2$ as the linear algebra constant and extrapolating from there, we set the number of rounds such that the attack becomes more expensive than using brute-force.

Our approach to Gröbner bases and their impact on security differs from other discussions appearing in the literature.

- For some ciphers, the Gröbner basis comes for free because the system of polynomial equations that models the attack is already a Gröbner basis. For instance, the *Flurry* and *Curry* ciphers [BPW06a] were designed to exhibit this property, but it was also observed in *MiMC* [ACG+19] and even AES [BPW06b]. In these cases, the security of the cipher is argued not based on the difficulty of (i) finding a Gröbner basis, but based on (ii) the difficulty of term order conversion.

- After publishing a pre-print version of this paper, our design principle (*i.e.*, relying on the difficulty of step (i) has been adopted by the *HadesMiMC* family of arithmetization-oriented ciphers [GLR+19]. However, in this case the system of polynomial equations arising from modeling an attack is assumed to be (semi-)regular, but this assumption is never experimentally tested.

- Before said publication, Gröbner basis algorithms had been used in the design of some ciphers [BBK14], but only as a tool to rule out flawed design strategies and never as the most important attack according to which the parameters are to be set.

- Gröbner basis analysis is a common theme in the design of post-quantum cryptosystem. However, the polynomial systems there are typically overdetermined and defined over small fields. As a result, steps *(ii)* and *(iii)* are obsolete because a solution can be read out from the Gröbner basis in any order, including *degrevlex*.

There are many ways to employ Gröbner basis algorithms in an attack. The particular flavor of the Gröbner basis attack we aim to resist is a preimage search for the arithmetic sponge-based hashing mode described in Section 4.3. In the experiments we model the absorption of one field element and extraction ("squeeze") of the same. Other flavors of the attack induce the same or greater complexity: a different trade-off between rate and capacity would result in variables being removed from the output side and added on the input side, and equations which are removed from the input side and placed on the output side. Overall, the number of variables and equations remains the same and the attack complexity does not change. Attempting a key recovery attack on the block cipher would require modeling also the key schedule algorithm, effectively doubling the size of the polynomial system. Likewise, calling the permutation twice in order to either absorb or squeeze another data block would again double the size of the polynomial system.

In determining the concrete observed degree of regularity we discovered an interesting inverse trade-off between the width of the state m and the number of rounds N required to resist the Gröbner basis attack. We see that for higher m, the degree of regularity grows faster in each round.

4.3 Arithmetic Sponges

A sponge construction generates a hash function from an underlying permutation by iteratively applying it to a large state [BDPA08]. Usually, 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 absorption phase, r bits of the input are injected into the state until there is no more input to absorb; in every iteration of the squeezing phase, r bits of the state are read out until the desired output length is met. This definition is adapted to work over field elements, see [BDPA07]. Instead of working over bits, the arithmetic rate now consists of r_q field elements in $\mathbb{F}_q$. The remaining $c_q = m - r_q$ field elements are said to be the arithmetic capacity of the sponge.

4.3.1 *Marvellous* sponges

To turn a *Marvellous* keyed algorithm into an unkeyed permutation, fix the secret key to zero. The resulting permutation can then be used in a sponge construction to obtain a *doubly-extendable output (DEC)* function, with hash functions as a private case when the output length is fixed.

A sponge function instantiated with the resulting permutation, arithmetic rate r_q, and arithmetic capacity c_q absorbs (using field addition) or squeezes up to r_q field elements per permutation-call. Note that increasing r_q and keeping c_q fixed effectively improves the throughput of the sponge function without significantly affecting the cost and not at all the security due to the trade-off discussed in Section 4.2.3. The generic security of a sponge function in this setting is $\log_2(\sqrt{q}) \min(c_q, r_q)$ bits of security assuming that the underlying permutation is randomly selected. It is common practice to consider a specific permutation as being randomly selected if there is no known distinguisher against it.

4.3.2 Padding

In case the input to the sponge is of variable length, the sponge model also requires that a padding rule be defined. We suggest the following rule: first, append to the end of the input the unit element $1 \in \mathbb{F}_q$ and then, if necessary, append as many zeros $0 \in \mathbb{F}_q$ as needed until the number of field elements in the input is divisible by r_q.

4.4 Efficiency

Throughout the *Marvellous* design, we only use arithmetization-efficient maps or their functional inverses. The realization of functional inverse maps is not always efficient when implemented straightforwardly. However, all use cases targeted by this paper enable non-procedural computations making it arithmetization-efficient to calculate the functional inverse of an arithmetization-efficient map.

As an instructive example, consider a field $\mathbb{F}_q$ satisfying $\gcd(q-1,\alpha)=1$ and $\alpha=3$ (*i.e.*, cubing is a permutation) and the set

$$\Lambda = \{(x,y)|y \text{ is the cube root of } x\}\,.$$

For a given pair $(x_0,y_0)\in\mathbb{F}_q\times\mathbb{F}_q$, checking if $(x_0,y_0)\in\Lambda$ can be done by calculating $x_0^{1/3}=x_0^{(2q-1)/3}$ and comparing the result to y_0; alternatively, by calculating y_0^3 and comparing the result to x_0. The former method has a multiplicative complexity of about $\log_2((2q-1)/3)$. This is much higher than $\log_2(3)$, the multiplicative complexity of the latter method. Using *Marvellous* designs mostly makes sense in settings where similar tools are afforded by the advanced cryptographic protocol. On the other hand, when such tools are available, our designs can efficiently employ operations that would have otherwise been discarded as being inefficient.

A particularly useful property of *Marvellous* designs is the counter-intuitive inverse trade-off between m and N (*i.e.*, the number of field elements in the state and the number of rounds, respectively) described in Section 4.2.3. This trade-off allows to treat m as a parameter that can be tweaked in order to improve the throughput of a sponge function or reduce the multiplicative depth of an instance in exchange for a larger state size. For example, a large m can be used to build an n-ary Merkle-tree rather than a binary one to reduce the tree depth (and hence the number of hashes per path) at the expense of larger inclusion proofs. As all the S-boxes in a single S-box layer operate in parallel, a large m also allows to compress more S-boxes into a single communication round in an MPC protocol.

5 *Vision*

We now describe the first *Marvellous* family, *Vision*. Since most aspects of the design are directly derived from the design strategy discussed in Section 4, we limit ourselves here to *Vision*-specific design decisions and provide the technical specification of the algorithm. In Appendix D, we provide two instances of the family. Additional instances can be generated using the Sage code on GitHub [SDS19].

Vision is meant to operate on binary fields with its native field $\mathbb{F}_{2^n}$, *i.e.*, $q=2^n$. The state is viewed as a column vector of m field elements and is an element of the vector space $\mathbb{F}_{2^n}^m$. To construct the S-boxes we first select an $\mathbb{F}_2$-linearized affine polynomial of degree 4 which we denote by B. Then,

$$\pi_1:\mathbb{F}_{2^n}\to\mathbb{F}_{2^n}: x\mapsto B(x^{-1})\,,$$

and

$$\pi_0:\mathbb{F}_{2^n}\to\mathbb{F}_{2^n}: x\mapsto B^{-1}(x^{-1})\,.$$

Advice on how to choose the $\mathbb{F}_2$-linearized affine polynomial, the step constants, and the MDS matrix was provided in Section 4.1.

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. Users who wish to use *Vision* as an unkeyed primitive in a sponge construction are referred to Section 4.3. The round function is depicted in Figure 1 and the pseudo-code for the cipher is given in Algorithm 1. The key schedule is obtained by feeding Algorithm 1 with the master key K as plaintext (*i.e.*, $P=K$) and the round constants as subkeys (*i.e.*, $K_s=C_s$).

For a desired security level of s, in bits, we set ℓ_0 as the number of rounds covered by the longest reaching attack in Table 1, $\ell_1=\lceil\frac{s+m+8}{8m}\rceil$ the number of rounds which can

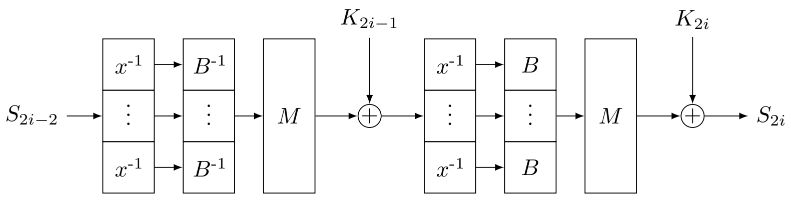


Figure 1: A single round (two steps) of *Vision*

be attacked by a Gröbner basis attack,[11] and 5 a sanity factor ensuring that aggressive optimization attempts do not result in trivially weak instances.

Finally, the number of rounds is set to $N = 2 \cdot \max(\ell_0, \ell_1, 5)$, *i.e.*, we take the longest reaching attack and add a 100% safety margin by doubling the number of rounds.

Algorithm 1: *Vision*

```
Input: Plaintext $P$, step keys $K_s$ for $0 \leq s \leq 2N$
Output: C = Vision $(K, P)$
    $S_0 = P + K_0$
    for $r = 1$ to $N$ do
        for $i = 1$ to $m$ do
            $Inter_r[i] = (S_{r-1}[i])^{-1}$
            $Inter_r[i] = B^{-1}(Inter_r[i])$
        end
        for $i = 1$ to $m$ do
            $S_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[j] + K_{2r-1}[i]$
        end
        for $i = 1$ to $m$ do
            $Inter_r[i] = (S_r[i])^{-1}$
            $Inter_r[i] = B(Inter_r[i])$
        end
        for $i = 1$ to $m$ do
            $S_r[i] = \sum_{j=1}^{m} M[i,j] Inter_r[j] + K_{2r}[i]$
        end
    end
    return $S_N$
```

5.1 Resistance to Gröbner Basis Attacks

The resistance of *Vision* against most attacks can be derived from Table 1. In this section we focus only on resistance to Gröbner Basis attacks in the preimage search setting as other settings result in higher effort for the adversary as explained in Section 4.2.3.

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-1} and K_{2i} represent known fixed values coming from the key injections. Furthermore, when isolated, $[m]$ denotes 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

[11]This bound is derived from Equation (8). A previous version of this paper used a different bound here, resulting in more rounds than what is strictly necessary. We decided to update the bound in this version since we are not familiar with any actual instance of *Vision* already in deployment. The authors apologize for the typo.

particular $M^{-1}[i,j]$ takes the (i,j)-th element of M^{-1}.

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

Note that left hand side of the second line is a polynomial in $S_{2i-1}[j]$ as the negative powers are canceled by the factor $(S_{2i+1}[j])^4$ and as the degree of the affine polynomial B is 4.

Analyzing the number of equations we see that the first step is described by r_q equations of degree 5 and c_q equations of degree 4.[12] The last step requires r_q equations of degree 4 and c_q equations of degree 5. Each of the other $2(N-1)$ steps requires m equations of degree 5. In total, the system is modeled by $r_q+c_q+2m(N-1)+r_q+c_q=2mN$ equations in $2mN$ variables.

Had the system been regular, its degree of regularity would have been given by the Macaulay bound

$$\begin{aligned} d_{\text{reg}} = 1 &+ \underbrace{\sum_{i=1}^{r_q}(\mathsf{deg}(f_i)-1)+\sum_{i=r_q+1}^{m}(\mathsf{deg}(f_i)-1)}_{\text{equations from the first step}} \\ &+ \underbrace{\sum_{j=m+1}^{m+2m(N-1)}(\mathsf{deg}(f_j)-1)}_{\text{equations excluding the first and last steps}} \\ &+ \underbrace{\sum_{\ell=m+2m(N-1)+1}^{m+2m(N-1)+r_q}(\mathsf{deg}(f_\ell)-1)+\sum_{\ell=m+2m(N-1)+r_q+1}^{2mN}(\mathsf{deg}(f_\ell)-1)}_{\text{equations from the last step}} \\ &= 1+4r_q+3c_q+4\cdot 2m(N-1)+3r_q+4c_q = 1+8mN-m\,. \end{aligned}$$

However, as we see below, the experimental result we have suggests that the concrete degree of regularity is actually smaller and we use the bound $\frac{d_{\text{reg}}}{4}\le d_{\text{con}}$ in determining a safe number of rounds.

5.1.1 Experimental Verification

We apply the first step of the attack, *i.e.*, finding a Gröbner basis in degree reverse lexicographic (degrevlex) order on round-reduced versions of *Vision*. We tried the attack on different algebra systems and using different libraries, with F_4 on Magma performing best.

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 only have a single data point after running the experiment for more than 60 hours. This data point has concrete degree of regularity 9, for $m=2$ and $N=2$ rounds.

We estimate the complexity of constructing a Gröbner basis in degree reverse lexicographic order to be at least

$$\binom{(1+16mN-m)/4}{2mN}^2, \tag{7}$$

[12] A more recent analysis showed that it is possible to completely avoid the first step. After much consideration the authors decided to absorb this improvement into the safety margin to retain consistency with the case of *Rescue*.

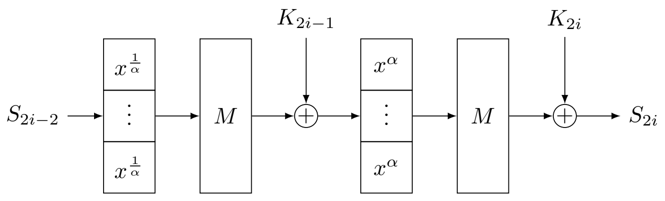


Figure 2: One round (two steps) of *Rescue* where the addition with the key is taken over a prime field.

and the number of rounds that can be attacked is calculated as

$$\ell_1 = \min(N) \quad \text{subject to} \quad \binom{(1+16mN-m)/4}{2mN}^2 \geq 2^s \,. \tag{8}$$

While it is risky to argue for a constant concrete-to-regular ratio from a single datum, we do have data in support of a similar conjecture coming from the experimental analysis of *Rescue*.

6 *Rescue*

The second family of algorithms in the *Marvellous* universe is *Rescue*. *Rescue* is similar to *Vision* in that it operates on field elements in $\mathbb{F}_q$, but this time q is prime rather than being a power of 2. In Appendix E, we provide several instances of *Rescue* with parameter sets which were derived from real world scenarios. Additional instances can be generated using the Sage code on GitHub [SDS19].

Again, the state is viewed as a column vector of m field elements and is an element of the vector space $\mathbb{F}_q^m$. To build the S-boxes we first find the smallest prime α such that $\gcd(q-1,\alpha)=1$. Whenever possible we recommend to choose the field such that $\gcd(q-1,3)=1$ as $\alpha=3$ was observed to result in the most efficient design. Then, the S-boxes are set to be $\pi_0 : \mathbb{F}_q \to \mathbb{F}_q : x \mapsto x^{1/\alpha}$ and $\pi_1 : \mathbb{F}_q \to \mathbb{F}_q : x \mapsto x^{\alpha}$. The MDS matrix and the round constants are generated per Section 4.1.

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. 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. As in the case of *Vision*, the key schedule is obtained by feeding Algorithm 2 with the master key K as plaintext (*i.e.*, $P=K$) and the step constants as subkeys (*i.e.*, $K_s=C_s$).

For a desired security level of s, in bits, we set ℓ_0 to be the number of rounds covered by the longest reaching attack in Table 1, $\ell_1 = \begin{cases} \lceil \frac{s+2}{4m} \rceil & \text{for } \alpha=3 \\ \lceil \frac{s+3}{5.5m} \rceil & \text{for } \alpha=5 \end{cases}$ the number of rounds which can be attacked by a Gröbner basis attack,[13] and 5 for the same reason as in *Vision*.

Finally, the number of rounds is set to $N = 2\cdot\max(\ell_0,\ell_1,5)$, *i.e.*, we take the longest reaching attack and add a 100% safety margin by doubling the number of rounds.

[13]These bounds are derived from Equation (9). Users interested in instances where $\alpha > 5$ should use the same number of rounds as in the case of $\alpha = 5$ if optimizing for security or derive ℓ_1 directly from Equation (9) if optimizing for performance.

Algorithm 2: *Rescue*

```
Input: Plaintext $P$, round keys $K_s$ for $0 \leq s \leq 2N$
Output: Rescue $(K, P)$
    $S_0 = P + K_0$
    for $r = 1$ to $N$ do
        for $i = 1$ to $m$ do
            $Inter_r[i] = K_{2r-1}[i] + \sum_{j=1}^{m} M[i,j](S_{r-1}[j])^{1/\alpha}$
        end
        for $i = 1$ to $m$ do
            $S_r[i] = K_{2r}[i] + \sum_{j=1}^{m} M[i,j](Inter_r[j])^{\alpha}$
        end
    end
    return $S_N$
```

6.1 Resistance to Gröbner Basis Attacks

Similar to *Vision*, the resistance of *Rescue* against most attacks can be derived from Table 1. Here we focus only on the resistance to Gröbner Basis attacks in the preimage search setting as other settings result in higher effort for the adversary as explained in Section 4.2.3.

Like for *Vision*, we provide equations encoding the preimage of the *Rescue* sponge function in the setting described in Section 4.2.3. In contrast to *Vision* it is now possible to fold equations across two steps in order to reduce the number of variables and equations.[14] As before we use $[m]$ to denote $\{1, \dots, m\}$ unless the brackets are a suffix to a vector or matrix, in which case the indicated element is meant.

$$\left\{ \begin{array}{ll} \left(\sum_{k=1}^{m} M^{-1}[j,k](S_1[k] - K_1[k]) \right)^{\alpha} - S_0[j] - K_0[j] & i = 1, j \in [m] \\ \left(\sum_{k=1}^{m} M[j,k] S_{2i-1}[k]^{\alpha} \right) + K_{2i}[j] - \left(\sum_{k=1}^{m} M^{-1}[j,k](S_{2i+1}[k] - K_{2i+1}[k]) \right)^{\alpha} = 0 & \\ & i \in [N-1], j \in [m] \\ \left(\sum_{k=1}^{m} M[j,k] S_{2N-1}[k]^{\alpha} \right) + K_{2N}[j] - S_{2N}[j] = 0 & i = N, j \in [r_q] \end{array} \right\}$$

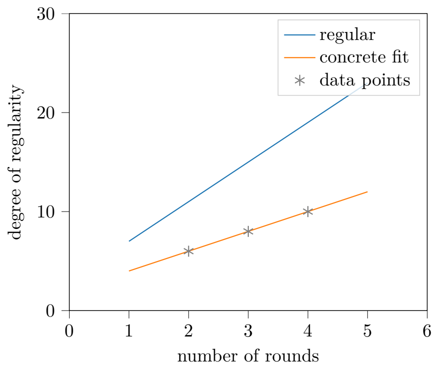
The first step introduces m equations of degree α in $r_q + m$ variables where the first r_q variables represent the unknown input and m variables the output of the step.[15] Then, every two subsequent steps introduce m equations of degree α in m new variables. The last step adds another r_q equations without introducing additional variables. In total, the model uses $r_q + mN$ equations of degree α in $r_q + mN$ variables.

If the system of equations were regular we would find via the Macaulay bound $d_{\text{reg}} = 1 + \sum_{i=1}^{mN+r_q} (\mathsf{deg}(f_i) - 1) = 1 + (mN + r_q)(\alpha - 1)$. Experimentally, for small round numbers we observe the concrete degree of regularity fits $d_{\text{con}} = 0.5mN(\alpha - 1) + 2$ which can be bounded as $\frac{d_{\text{reg}}}{2} \leq d_{\text{con}}$ for $\alpha \in \{3, 5\}$.

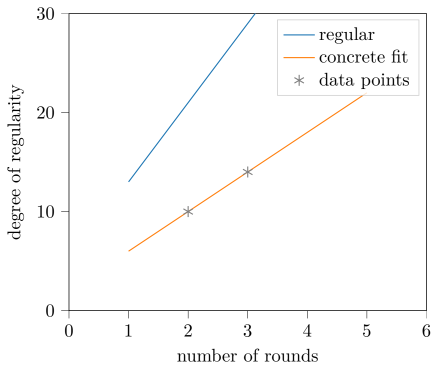
In Figure 3 we extrapolate the complexity of constructing a degree reverse lexicographic Gröbner basis of round reduced versions of *Rescue* for different m and α assuming the same concrete-to-regular degree ratio holds even for larger round numbers. For comparison, we also depict the expected complexity if the system were random.

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

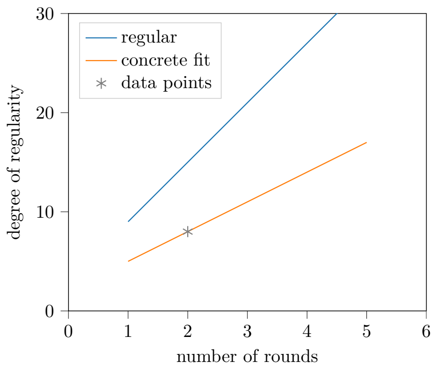
[15] Similar to the case of *Vision*, a more recent analysis showed that the first step can be avoided. The authors decided to absorb this improvement into the safety margin to avoid inconsistencies with already deployed versions.



(a) $m = 2, \alpha = 3$



(b) $m = 3, \alpha = 3$



(c) $m = 2, \alpha = 5$

Figure 3: Concrete degree of regularity for instances of *Rescue* (blue) compared to the expected degree of regularity for a random polynomial system of the same number of equations and variables (orange). In all cases, the experimental data fits $0.5(\alpha - 1)mN + 2$ suggesting a concrete-to-regular degree ratio $\frac{d_{\text{reg}}}{2} \leq d_{\text{con}}$.

6.1.1 Experimental Verification

We calculated the degree of the Gröbner basis output by the Gröbner basis algorithm for several round-reduced versions of *Rescue* in a sponge construction. Since the number of variables and the number of equations both depend on r_q, different rate values would lead to different complexities for the attack. In all our experiments we used $r_q = 1$ as this case represents the simplest model, *i.e.*, the best case scenario from the adversary's point of view.

The exact points we get for $m = 2, \alpha = 3$ are $\{(N, d_{\text{con}}) | (2, 6), (3, 8), (4, 10)\}$, for $m = 3, \alpha = 3$ we have the point $(N, d_{\text{con}}) = (2, 8)$, and for $m = 2, \alpha = 5$ we have the points $\{(N, d_{\text{con}}) | (2, 10), (3, 14)\}$. Observing that these points are fitted by $0.5mN(\alpha - 1) + 2$ we estimate the complexity of finding the first Gröbner basis by

$$\binom{mN(0.5(\alpha - 1) + 1) + r_q + 2}{mN + r_q}^2,$$

and the number of rounds that can be attacked can be determined according to

$$\ell_1 = \min(N) \quad \text{subject to} \quad \binom{mN(0.5(\alpha - 1) + 1) + r_q + 2}{mN + r_q}^2 \geq 2^s. \tag{9}$$

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 *Starkad* and *Poseidon* [GKK+19], and $GMiMC_{erf}$ [AGP+19].

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 $[i, j]$ (resp., $[i]$) 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, *e.g.*, $\{a_i, b_i \mid i \in [2]\} = \{a_1, a_2, b_1, b_2\}$.

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 [BBHR18] we recall those, along with a simple motivating example in Appendix B.

7.1.1 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* 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 $C_k[i] \in \mathbb{F}_{2^n}$ be the i-th field element of the k-th step constant, and let $B(Z) = b_0 + b_1 Z + b_2 Z^2 + b_3 Z^4$ be the $\mathbb{F}_2$-linearized affine 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 $K'[i] = K[i]^{-1}$ due to the first constraint, 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 a single step of the key schedule, with the main difference being that instead of adding a step constant (denoted C_k above) we add

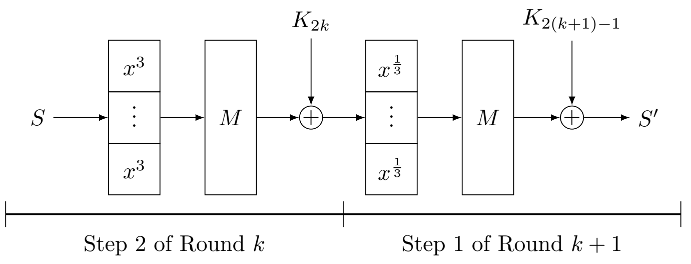


Figure 4: An adapted representation of a round of *Rescue* better suited for STARK evaluation.

the k-th 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.

Vision can be transformed to a permutation and used in a sponge construction where the keys are fixed to certain known constants. The key schedule is dropped, leading to an AIR of width $\mathsf{w} = 2m$ and $\mathsf{t} = 2$ cycles per step.

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

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

Rescue is quite similar to *Vision* but simpler from an algebraic perspective. The 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 non-procedural computation. Consider an adapted round as shown in Figure 4. Here, the first step of the adapted round is "folded" into its second step. This leaves the first and last steps of the algorithm 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 $S'[1], \dots, S'[m]$ hold the correct values, given $S[1], \dots, S[m]$, $K_{2k-1}[1], \dots, K_{2k-1}[m]$ and $K_{2(k+1)-1}[1], \dots, K_{2(k+1)-1}[m]$,

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

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 requires $\mathsf{t} = 1$ cycles 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.

7.2 Zero-Knowledge Proofs Based on R1CS Systems

In this section we evaluate the efficiency of *Vision* and *Rescue* 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 [PGHR13], ZK-SNARK [BCG+13], Aurora [BCR+19], Ligero [AHIV17], and Bulletproofs [BBB+18].

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

Recalling the two cycles of the AIR for *Vision* recounted earlier for constructing each of the key and round (Section 7.1.1), 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

$$\sum_j M[i,j]b_3R[j]^2 = (\sum_j \alpha_j R[j])^2$$

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).

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

To efficiently encode a step of *Rescue* for $\alpha = 3$, we use two R1CS constraints to compute the cube of a state variable giving a total of $2m$ constraints for the cubing operations over the whole state. The step using the inverse cube 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. We consider three masked operation techniques: one technique to find the inverse of a shared field element due to Bar-Ilan and Beaver [BB89]; one technique to raise a shared element to an arbitrary but known power due to Damgård *et al.* [DFK+06]; and one novel technique to compute the compositional inverse of a low-degree linearized polynomial. Their descriptions can be found in Appendix C.

The common strategy behind these techniques is to apply random unknown masks to a shared secret value and opening their sum. The operation proper is applied to the opened variable. However, due to the variable still being masked, the opened value does not leak information on the secret. 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 offloading 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.

7.3.1 Computing a *Vision* Round over MPC

Recall that elements of the state in *Vision* are members of the extension field $\mathbb{F}_{2^n}$. 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.

To evaluate the inversion step, we use the technique from Bar-Ilan and Beaver [BB89], of which pseudocode is given in Appendix C. In scenarios where the shared value is unlikely to be zero (*e.g.*, 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 C.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

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

7.3.2 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.* [DFK+06]. This way, we can offload a portion of the computation to an offline phase and retain a constant online complexity (*i.e.*, 1 communication round). A small adaptation of this technique computes the inverse power map at the same online cost. We summarize this adaptation in Appendix C.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

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

7.4 Comparison as Hash Functions

This section discusses the performance of our primitives *Vision* and *Rescue*. In particular, we evaluate and compare our primitives in the sponge construction as hash functions. This way, we can use the parameters for the comparison from the "STARK-Friendly Hash Challenge" [Stab] as they are representative for practical applications and of contemporary interest to both industry and academia (see *e.g.*, [BCD+20]). So far, all the applications we are aware of (including the SFH challenge) are interested in *Marvellous* hash functions which is yet another reason to use them as the focal point for the comparison here. We discard the 40-bit security level puzzles of the challenge, and only consider 80, 128, and 256 bits of security using different parameter sets for these security levels. As the main difference between the block cipher and hash function is the key schedule, the cost of evaluating *Vision* and *Rescue* as block ciphers would stay the same or double whether or not the key schedule can be pre-computed. The implementation of the instances we consider can be found in the form of Sage code in [Stab].

We compare our algorithms *Vision* and *Rescue* with *Starkad* and *Poseidon* [GKK+19], and *GMiMC*$_{erf}$ [AGP+19] as these ciphers are also optimized towards minimizing their algebraic representations to speed up advanced cryptographic protocols. We stress that while the nominal figures we provide serve as a comparison point for the respective efficiencies, they completely overlook the confidence aspect, *i.e.*, that by design, *Vision* and *Rescue* employ much larger safety margins, compared to *Starkad*, *Poseidon*, and *GMiMC*$_{erf}$ and are thus more robust against advances in the cryptanalysis of arithmetization-oriented algorithms.

For the purpose of the present comparison, the AIR cost is given by the value of $\mathtt{w} \cdot \mathtt{t} \cdot \mathtt{d}$. In the MPC comparison we ignore the zero-test and observe that the offline parts can be executed in parallel for all rounds. We compare the algorithms as sponge functions for AIR (Table 2), R1CS (Table 3), and masked MPC (Table 4). We recall that *Vision* requires $2\lceil (s + m + 8)/8m \rceil$ rounds and *Rescue* requires $2\lceil (s + 2)/4m \rceil$ rounds for $\alpha = 3$, both with a minimum of 10 rounds, where s denotes the security level in bits, and m denotes the number of state elements. We stress that the field size does not change the cost under the metrics we consider in this paper (*i.e.*, arithmetic complexity).

It can be seen in Table 2 that the AIR description of *Rescue* is significantly more efficient than the other candidates in virtually all parameter sets. In Table 4 we see that for the MPC case *Rescue* always outperforms the other algorithms in terms of online communication rounds, with *Vision* and *Poseidon* competing for the leadership in terms of number of multiplications. This is partly due to the larger safety margins and partly due to different optimization strategies: while *Starkad* and *Poseidon* are optimized to minimize the number of field multiplications, *Vision* and *Rescue* are optimized to minimize the number of communication rounds (*i.e.*, circuit depth). Finally, in Table 3 we see that in most cases *Poseidon* outperforms the other candidates. This result is mostly due to the difference in the safety margins and the trend can be expected to be different after normalization. Nevertheless, a comparison with Table 2 highlights that it is possible to optimize an algorithm towards one proof system and not the other.

Note that due to the flexible nature of these algorithms, it is difficult to directly compare them to existing algorithms such as AES and SHA2. For completeness, we refer the reader to [BBHR18, Fig. 4] with the disclaimer that the results therein are not an apples-to-apples comparison and are in fact inferior in terms of both security and throughput. Still, even the most efficient choice there (*i.e.*, AES-128 in Davies-Meyer) induces a cost of $\mathsf{w} \cdot \mathsf{c} \cdot \mathsf{d} = 62 \cdot 48 \cdot 8 = 23{,}808$ while the worst option in Table 2 (*i.e.*, *Vision* in the last row) is about 21 times more efficient. This serves to show the usefulness of algebraic ciphers compared to traditional ones in certain settings. A recent work by Bonte *et al.* compared the running times of threshold variants of some NIST standardized signature schemes in MPC and observed that using *Rescue* is 200-350 faster compared to using their native SHA2-512/SHAKE-256 hashes. For more details, see [BST20, Tab. 2–3].

Table 2: Comparison of *Vision*, *Rescue*, *Starkad*, *Poseidon*, and *GMiMC$_{erf}$* over AIR where ℓ denotes the bit-size of the base field, c denotes the capacity of the sponge in terms of field elements and r its rate. The efficiency is given by $\mathsf{w} \cdot \mathsf{t} \cdot \mathsf{d}$ which is the product between the length and the width of the trace together with the degree of the constraints.

Parameters				*Vision*	*Rescue*	*Starkad*	*Poseidon*	*GMiMC$_{erf}$*
80 bit security								
$\ell \approx 80$	$m = 4$	$c = 2$	$r = 2$	320	**156**	255	249	333
$\ell \approx 160$	$m = 3$	$c = 1$	$r = 2$	240	**135**	228	222	630
$\ell \approx 160$	$m = 11$	$c = 1$	$r = 10$	880	**363**	426	420	678
128 bit security								
$\ell \approx 128$	$m = 4$	$c = 2$	$r = 2$	320	**228**	351	345	498
$\ell \approx 256$	$m = 3$	$c = 1$	$r = 2$	288	**207**	327	321	978
$\ell \approx 128$	$m = 12$	$c = 2$	$r = 10$	960	**396**	552	543	546
$\ell \approx 64$	$m = 12$	$c = 4$	$r = 8$	960	396	417	411	**303**
$\ell \approx 256$	$m = 11$	$c = 1$	$r = 10$	880	**363**	528	519	1026
256 bit security								
$\ell \approx 128$	$m = 8$	$c = 4$	$r = 4$	640	**456**	450	444	522
$\ell \approx 128$	$m = 14$	$c = 4$	$r = 10$	1120	**462**	600	591	558

Table 3: Comparison of *Vision*, *Rescue*, *Starkad*, *Poseidon*, and *GMiMC$_{erf}$* over R1CS where ℓ, c, and r are as in the case of Table 2. The efficiency is given by the overall number of constraints.

Parameters				*Vision*	*Rescue*	*Starkad*	*Poseidon*	*GMiMC$_{erf}$*
80 bit security								
$\ell \approx 80$	$m = 4$	$c = 2$	$r = 2$	400	192	170	**166**	222
$\ell \approx 160$	$m = 3$	$c = 1$	$r = 2$	300	168	152	**148**	420
$\ell \approx 160$	$m = 11$	$c = 1$	$r = 10$	1100	440	284	**280**	452
128 bit security								
$\ell \approx 128$	$m = 4$	$c = 2$	$r = 2$	400	288	234	**230**	332
$\ell \approx 256$	$m = 3$	$c = 1$	$r = 2$	360	264	218	**214**	652
$\ell \approx 128$	$m = 12$	$c = 2$	$r = 10$	1200	480	368	**362**	364
$\ell \approx 64$	$m = 12$	$c = 4$	$r = 8$	1200	480	278	274	**202**
$\ell \approx 256$	$m = 11$	$c = 1$	$r = 10$	1100	440	352	**346**	684
256 bit security								
$\ell \approx 128$	$m = 8$	$c = 4$	$r = 4$	800	576	300	**296**	348
$\ell \approx 128$	$m = 14$	$c = 4$	$r = 10$	1400	560	400	394	**372**

Table 4: Comparison of *Vision*, *Rescue*, *Starkad*, *Poseidon*, and *GMiMC$_{erf}$* over MPC where ℓ, c, and r are as defined for Table 2. The efficiency is given in the number of online communication rounds (R.) and number of field multiplications (Mult.).

Parameters				*Vision*		*Rescue*		*Starkad*		*Poseidon*		*GMiMC$_{erf}$*	
80 bit security				R.	Mult.	R.	Mult.	R.	Mult.	R.	Mult.	R.	Mult.
$\ell \approx 80$	$m = 4$	$c = 2$	$r = 2$	50	280	**24**	384	61	255	59	**249**	111	333
$\ell \approx 160$	$m = 3$	$c = 1$	$r = 2$	50	**210**	**28**	336	60	228	58	222	210	630
$\ell \approx 160$	$m = 11$	$c = 1$	$r = 10$	50	770	**20**	880	62	426	60	**420**	226	678
128 bit security													
$\ell \approx 128$	$m = 4$	$c = 2$	$r = 2$	50	**280**	**36**	576	93	351	89	345	166	498
$\ell \approx 256$	$m = 3$	$c = 1$	$r = 2$	60	**252**	**44**	528	93	327	91	321	326	978
$\ell \approx 128$	$m = 12$	$c = 2$	$r = 10$	50	840	**20**	960	94	552	91	**543**	182	546
$\ell \approx 64$	$m = 12$	$c = 4$	$r = 8$	50	840	**20**	960	51	417	48	411	101	**303**
$\ell \approx 256$	$m = 11$	$c = 1$	$r = 10$	50	770	**20**	880	96	528	93	**519**	342	1026
256 bit security													
$\ell \approx 128$	$m = 8$	$c = 4$	$r = 4$	50	560	**36**	1152	94	450	90	**444**	174	522
$\ell \approx 128$	$m = 14$	$c = 4$	$r = 10$	50	980	**20**	1120	94	600	91	591	186	**558**

Acknowledgments

The authors would like to thank Vincent Rijmen and Daira Hopwood for their useful comments. Additionally, we want to thank the anonymous reviewers and Brice Minaud for sheparding the work.

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 and by an FWO post-doctoral fellowship under Grant Number 12ZH420N. 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

[ACG$^+$19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of stark-friendly designs: Application to marvellous and mimc. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, pages 371–397, 2019.

[AD18] Tomer Ashur and Siemen Dhooghe. Marvellous: a stark-friendly family of cryptographic primitives. *IACR Cryptology ePrint Archive*, 2018:1098, 2018.

[AGP$^+$19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for mpc, and more. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, pages 151–171, 2019.

[AGR$^+$16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT 2016, Part I*, LNCS, pages 191–219, 2016.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM - CCS 2017*, October 2017.

[ARS$^+$15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 430–454, 2015.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *ACM Symposium on Principles of Distributed Computing 1989*, pages 201–209, 1989.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018.

[BBC$^+$17] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 551–579, 2017.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2014.

[BCD$^+$20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. *IACR Cryptology ePrint Archive*, 2020:188, 2020.

[BCF$^+$16] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *CoRR*, abs/1610.03798, 2016.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*, LNCS, pages 90–108, 2013.

[BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *TCC 2016*, LNCS, pages 33–64, 2016.

[BCL+20] Tim Beyne, Anne Canteaut, Gregor Leander, María Naya-Plasencia, Léo Perrin, and Friedrich Wiemer. On the security of the rescue hash function. *IACR Cryptol. ePrint Arch.*, 2020:820, 2020.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019.

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, 2007. https://keccak.team/files/SpongeFunctions.pdf.

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT 2008*, pages 181–197, 2008.

[Ben] Eli Ben-Sasson. The state of stark tooling. https://www.youtube.com/watch?v=UNbWFNdz95g.

[BFS04] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[BFS15] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 gröbner basis algorithm. *J. Symb. Comput.*, 70:49–70, 2015.

[BPW06a] Johannes A. Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. Block ciphers sensitive to gröbner basis attacks. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2006.

[BPW06b] Johannes A. Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. A zero-dimensional gröbner basis for AES-128. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 78–88. Springer, 2006.

[BST20] Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. Thresholdizing HashEdDSA: MPC to the rescue. *IACR Cryptology ePrint Archive*, 2020:214, 2020.

[CKM97] Stéphane Collart, Michael Kalkbrener, and Daniel Mall. Converting bases with the gröbner walk. *J. Symb. Comput.*, 24(3/4):465–469, 1997.

[CLO97] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.).* Undergraduate texts in mathematics. Springer, 1997.

[CMR06] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. *Algebraic aspects of the advanced encryption standard.* Springer, 2006.

[DFK$^+$06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC 2006*, pages 285–304, 2006.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[EGL$^+$20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. An algebraic attack on ciphers with low-degree round functions: Application to full mimc. *IACR Cryptology ePrint Archive*, 2020:182, 2020.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of pure and applied algebra*, 139(1-3):61–88, 1999.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (f_5). In *ISSAC 2002*, pages 75–83. ACM, 2002.

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[GKK$^+$19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. https://eprint.iacr.org/2019/458.

[GLR$^+$19] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. *IACR Cryptology ePrint Archive*, 2019:1107, 2019.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In *FSE 1997*, LNCS, pages 28–40, 1997.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In *CRYPTO (1)*, volume 9215 of *Lecture Notes in Computer Science*, pages 161–185. Springer, 2015.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Foundations of Computer Science 1990*, pages 2–10. IEEE, 1990.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for arx: Application to salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. https://eprint.iacr.org/2013/328.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Inscrypt*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[Nyb93] Kaisa Nyberg. Differentially uniform mappings for cryptography. In *EUROCRYPT 1993*, LNCS, pages 55–64, 1993.

[oS77] National Bureau of Standards. Data encryption standard. U.S. Department of Commerce, FIPS pub. 46, January 1977.

[PGHR13] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy 2013*, Oakland '13, pages 238–252, 2013.

[Raz87] Alexander A. Razborov. 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*, volume 41 - 4, pages 333–338, 1987.

[RDP$^+$96] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, pages 99–111, 1996.

[SDS19] Alan Szepieniec, Siemen Dhooghe, and Ferdinand Sauer. Marvellous (instance generator), 2019. https://github.com/KULeuven-COSIC/Marvellous.git.

[Staa] StarkWare Industries. STARK-friendly hash. Medium. https://medium.com/starkware/stark-friendly-hash-tire-kicking-8087e8d9a246.

[Stab] StarkWare Industries. Stark-friendly hash challenge. https://starkware.co/hash-challenge/.

[Wie86] Douglas Wiedemann. Solving sparse linear equations over finite fields. *IEEE transactions on information theory*, 32(1):54–62, 1986.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *STOC 2012*, pages 887–898. ACM, 2012.

[WSR$^+$15] Riad S. Wahby, Srinath Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS 2015*, LNCS, 2015.

A Gröbner Basis Attacks

We recall here some basic facts about attacking symmetric-key algorithms using Gröbner basis algorithms. For more general information on the underlying mathematics, we refer the reader to Cox *et al.* [CLO97]. 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.* [BPW06a].

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, provided we work with zero-dimensional ideals, 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 [FGLM93] and Gröbner Walk [CKM97] 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 [Fau99, Fau02]. In every iteration, F_4 extends 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 improves on this strategy by tracking the origin of all intermediate polynomials, enabling the algorithm to predict beforehand when a polynomial reduction will result in no new information. As a result, when applied to a worst-case class of polynomial systems called *regular* systems — systems that exhibit no non-trivial algebraic dependencies in the same sense that non-singular matrices exhibit no linear dependencies — the F_5 algorithm can be proven not to perform useless reductions before the step degree reaches the ideal's *degree of regularity* [BFS04, BFS15]. For regular systems, this degree of regularity is given by the Macaulay bound: $d_{\text{reg}} \leq 1 + \sum_{i=1}^{m}(\mathsf{deg}(f_i) - 1)$. For both regular and irregular systems, the degree of regularity is informally equal to the degree of the Gröbner basis (by which we mean the maximum degree of all polynomials in the basis) in a degree-refining order such as *degrevlex* (but not *lex*). In particular, this means that neither F_4 nor F_5 will terminate before reaching this degree.

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 $\mathrm{HS}(z) = \frac{\prod_{i=1}^{m}(1-z^{\mathsf{deg}(f_i)})}{(1-z)^n}$, 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_4 and F_5 must reach this degree before it terminates, for overdetermined systems 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 $\binom{n+d_{\text{reg}}}{n}$ monomials of degree d_{reg} or less, and $\binom{n+d_{\text{reg}}}{n}^{\omega}$ 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 [Wil12] are used; and $\omega = 2$ when sparse linear algebra techniques such as Wiedemann's algorithm [Wie86] 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 [BPW06a]. The bottleneck of the attack then consists of the monomial order conversion using either FGLM or Gröbner Walk.

In stark contrast to that of the ciphers analyzed by Buchmann *et al.*, 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.

B STARKs

This appendix provides a brief background on Scalable Transparent ARguments of Knowledge (STARKs) [BBHR18]. We start with a motivating example in Section B.1 serving to give the reader an intuition about the way STARKs operate, deferring more formal treatment to Section B.2.

B.1 Intuition

Scalable Interactive Oracle Proofs (IOPs) and Transparent Arguments of Knowledge (STARKs) like [BBC+17, BBHR18] 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 x_q is the q-th element in the Fibonnacci sequence defined recursively for $i > 1$ by $x_i = x_{i-1} + x_{i-2} \mod p$."

An execution trace *proving* the statement above is a $(q+1) \times 1$ array in which the i-th 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 i-th 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.

B.2 Formal Description of an Algebraic Execution Trace

We start with the definition of an algebraic execution trace.

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

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

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

We point out that the definition of AIR in [BBHR18] is slightly more complicated (dealing with boundary constraints and neighborhood sets) but for the purpose of the current work the simpler definition above suffices.

C Algorithms for Masked MPC

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

C.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;
}
```

C.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 offline rounds)
        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)
}
```

C.3 Alpha-power and Inverse Alpha-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 $(xr^\alpha)^{1/\alpha}[r^{-1}] = [x^{1/\alpha}rr^{-1}] = [x^{1/\alpha}]$. All techniques assume $x \neq 0$, n equality check is done at the start to ensure this. 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;
}
```

D Instances of *Vision*

In this appendix we provide two instances of *Vision*. A *Vision* object is instantiated by loading the Sage code from [SDS19], and calling

`vision_instance = Vision(s, nm, m ).`

with s the desired security level in bits, nm the state size, and m the number of field elements in the state. Given a *Vision* object, the MDS matrix, the first round constant, and the affine transformation to generate the subsequent round constants can be obtained by calling the functions in Table 5.

Table 5: Functions for obtaining the building block of a *Vision* instance

Function	Description
`vision_instance.MDS`	returns the MDS matrix
`vision_instance.initial_constant`	returns the first step constant
`vision_instance.constants_matrix`	returns the linear part of the affine transformation generating subsequent step constants
`vision_instance.constants_constant`	returns the fixed part of the affine transformation generating subsequent step constants
`vision_instance.B`	returns the F_2-affine linearized polynomial
`vision_instance.Binv`	returns the inverse of the F_2-affine linearized polynomial
`vision_instance.Nb`	returns the number of rounds

We encourage users to create their own instances that are optimized for their own use cases. The instances below can be used as a target for cryptanalysis.

D.1 *Vision* Mark I

The first instance we consider has parameter sets comparable to those of AES. We use a binary field $\mathbb{F}_{2^n}$ with $n = 8$, and $m = 16$ state elements. Instances providing for 128-, 192-, and 256-bit security can be generated by calling

```
vision_instance = Vision(s = 128, nm = 128, m = 16);
vision_instance = Vision(s = 192, nm = 128, m = 16);
vision_instance = Vision(s = 256, nm = 128, m = 16),
```

and their building blocks as well as the required number of rounds can be obtained by calling the commands we listed above.

D.2 *Vision* Mark II

The second instance uses $n = 128$ and $m = 4$ state elements in $\mathbb{F}_{2^{128}}$. This instance is generated by calling

```
vision_instance = Vision(s = 128, nm = 512, m = 4),
```

and its building blocks as well as the required number of rounds can be obtained by calling the commands we listed above. By using it inside a sponge construction with $r_q = 2$ this instance is suitable for hashing and provides 128-bit collision resistance.

E Instances of *Rescue*

We provide now three instances of *Rescue* based on real-world scenarios. As before, a *Rescue* object is generated by loading the code from [SDS19], and calling

```
rescue_instance = Rescue(s, q, m, alpha),
```

with s the desired security level in bits, q the field, m the number of field elements per round, and α the S-box power map. The MDS matrix, the first round constant, and the affine transformation to generate the subsequent round constants can be obtained by calling the functions in Table 6.

Table 6: Functions for obtaining the building block of a *Vision* instance

Function	Description
`rescue_instance.MDS`	returns the MDS matrix
`rescue_instance.initial_constant`	returns the first step constant
`rescue_instance.constants_matrix`	returns the linear part of the affine transformation generating subsequent step constants
`rescue_instance.constants_constant`	returns the fixed part of the affine transformation generating subsequent step constants
`rescue_instance.Nb`	returns the number of rounds

Again, users are encouraged to generate their own instances using the provided code and cryptanalysts are encouraged to evaluate their security.

E.1 *Rescue* Mark I

For the first instance of *Rescue* we take a look at a sponge construction with a parameter set chosen by StarkWare Industries at the end of [Stab].[16]

The field is $\mathbb{F}_q$ where

```
q = 2**(61) + 20 * 2**(32) + 1
```

and the state consists of m = 12 elements. In [Stab] the power map is chosen to be $\alpha = 3$, the rate is $r = 8$ with capacity $c = 4$, and the number of rounds is set to be $N = 10$. This provides 122 bits of security.

[16]The StarkWare challenge used a different MDS matrix than the one generated by our code. Cryptanalysts are invited to attack either of the two versions.

The instance is generated by calling

```
rescue_instance = Rescue(s = 122, q, m = 12, alpha = 3),
```

and its building blocks can be obtained by calling the commands we listed above.

E.2 *Rescue* Mark II

Our second *Rescue* instance uses a parameter set optimized for the HashEdDSA Ed25519 thresholdized signature scheme. This parameter set was evaluated in [BST20] and was shown to be 150–200 times faster than the same protocol when employing SHA2-512.

The field is $\mathbb{F}_{\texttt{q}}$ where `q` is the order of the curve Ed25519. We note that $\log_2(\texttt{q}) = 252$, or more specifically:

```
q = 2**(252) + 27742317777372353535851937790883648493.
```

The construction uses `m = 6`, $\alpha = 5$, rate $r = 4$ and capacity $c = 2$. With $N = 10$ this instance provides `s = 128` bits of security.

The instance is generated by calling

```
rescue_instance = Rescue(s = 128, q, m = 6, alpha = 5),
```

and its building blocks can be obtained by calling the commands we listed above.

E.3 *Rescue* Mark III

The third instance provides a parameter set for optimizing a sponge construction working over the curve Ed448. It was evaluated in [BST20] to be 135–355 times faster than the same protocol when employing SHAKE-256.

We use the prime field $\mathbb{F}_{\texttt{q}}$ where `q` is the order of the curve Ed448. We note that $\log_2(\texttt{q}) = 446$, or more specifically

```
q = 2**(446)-
13818066809895115352007386748515426880336692474882178609894547503885.
```

[BST20] uses a state consisting of `m = 10` field elements, with rate $r = 8$ and capacity $c = 2$. With $n = 10$ rounds this isntance provides 224 bits of security.

The instance is generated by calling

```
rescue_instance = Rescue(s = 224, q, m = 10, alpha = 5),
```

and its building blocks can be obtained by calling the commands we listed above.