Algebraic Attacks against Some Arithmetization-Oriented Primitives

Augustin Bariant¹, Clémence Bouvier¹,², Gaëtan Leurent¹, Léo Perrin¹

¹Inria, Paris, France
²Sorbonne University, Paris, France

Abstract. Recent advanced Zero-Knowledge protocols, along with other high-level constructions such as Multi-Party Computations (MPC), have highlighted the need for a new type of symmetric primitives that are not optimized for speed on the usual platforms (desktop computers, servers, microcontrollers, RFID tags...), but for their ability to be implemented using arithmetic circuits.

Several primitives have already been proposed to satisfy this need. In order to enable an efficient arithmetization, they operate over large finite fields, and use round functions that can be modelled using low degree equations. The impact of these properties on their security remains to be completely assessed. In particular, algebraic attacks relying on polynomial root-finding become extremely relevant. Such attacks work by writing the cryptanalysis as systems of polynomial equations over the large field, and solving them with off-the-shelf tools (SageMath, NTL, Magma, ...).

The need for further analysis of these new designs has been recently highlighted by the Ethereum Foundation, as it issued bounties for successful attacks against round-reduced versions of several of them.

In this paper, we show that the security analysis performed by the designers (or challenge authors) of four such primitives is too optimistic, and that it is possible to improve algebraic attacks using insights gathered from a careful study of the round function.

First, we show that univariate polynomial root-finding can be of great relevance in practice, as it allows us to solve many of the Ethereum Foundation’s challenges on Feistel–MiMC. Second, we introduce a trick to essentially shave off two full rounds at little to no cost for Substitution-Permutation Networks (SPN). This can be combined with univariate (resp. multivariate) root-finding, which allowed to solve some challenges for Poseidon (resp. Rescue–Prime). Finally, we also find an alternative way to set up a system of equations to attack Ciminion, leading to much faster attacks than expected by the designers.

Keywords: Arithmetization-oriented hash functions · Poseidon · Feistel–MiMC · Rescue–Prime · Ciminion · algebraic cryptanalysis

1 Introduction

Up until a few years ago, the vast majority of new symmetric primitives were optimized to run on the “usual” platforms, from high-end computers like desktops or servers with sophisticated processors handling e.g. vector and AES [AES01] instructions, all the way down to microcontrollers and RFID tags—the so-called lightweight cryptography. The design or the analysis of a primitive intended to run in such contexts will then be based on a vast literature, building upon decades of improvements of well-known techniques such as the differential attack [BS92, BS91].
Things are different in the arithmetization-oriented (AO) world. This term describes primitives that are not optimized for time and memory complexities on the usual platforms, but to yield an efficient representation as an arithmetic circuit, where the arithmetic operations considered are the addition and the multiplication in a large finite field \( F_q \), where \( q \) is typically at least equal to \( 2^{63} \), and is often a prime number. The subtleties of the arithmetization considered will impact the cost of each operation, and may in fact enable the use of more sophisticated operations (for example, Reinforced Concrete \([GKL+21]\) uses the ability of Plonk \([GWC19]\) to evaluate lookup tables).

As a first approximation, AO designs must be such that there exists a low degree model of their round function. Suppose that such a primitive operates on \( F_m^q \), for some \( m \geq 1 \), and that its round function is \( R : F_m^q \rightarrow F_m^q \). Then there must exist a family of \( m \) polynomials \( \{P_i\}_{0 \leq i < m} \) such that:

- \( P_i : (F_m^q)^2 \rightarrow F_q \),
- \( P_i(x, y) = 0 \) for all \( i \) if and only if \( y = R(x) \), and
- the multiplicative complexity of each \( P_i \) is low (for efficiency purposes).

A direct approach to reach this goal is to use a low degree round function, as was done in Feistel–MiMC \([AGR+16]\), Poseidon \([GKR+21]\), and Ciminion \([DGGK21]\). A more sophisticated approach was introduced by the designers of Rescue \([AAB+20, SAD20]\), who remarked that such a low degree encoding was also possible if the inverse of the round function had a low degree.

As a direct consequence of these design criteria, such primitives may be vulnerable to a specific form of cryptanalysis. Indeed, there exist off-the-shelf tools that are capable of finding the roots of a system of polynomials, and which are more efficient when the degree of the equations is lower. Such attacks have been lumped together and called algebraic attacks, but this simplification erases some important subtleties, as we will see.

Algebraic attacks are indeed a threat. For instance, Jarvis \([AD18]\) was found to be vulnerable to some algebraic attacks \([ACG+19]\): it turns out that the system of equations needed to model its round function envisioned by its designers could be greatly simplified in a systematic fashion. However, that is not to say that only such algebraic methods can be applied to such primitives. In fact, as shown in \([BCD+20]\), more classical distinguishers based on higher-order differentials and subspace trails can be applied to such primitives, namely the permutations GMiMC \([AGP+19]\), and Poseidon \([GKR+21]\). Subspace-based attacks against the latter were substantially improved in \([KR21]\). The initial security margin of the AO block cipher MiMC \([AGR+16]\) against higher-order differentials was also re-assessed twice, first by Eichlseder \textit{et al.} \([EGL+20]\), then by Bouvier \textit{et al.} \([BCP22]\).

The pressing need for a better understanding of the security of AO hash functions has pushed the Ethereum foundation to put forward a bounty\(^1\) rewarding with thousands of dollars the best practical attacks against round-reduced versions of the permutations underlying several sponge-based AO hash functions, namely Reinforced Concrete \([GKL+21]\), Feistel–MiMC \([AGR+16]\), Poseidon \([GKR+21]\), and Rescue–Prime \([AAB+20, SAD20]\). The specific parameter sets targeted at the time of publication of the bounties are specified in Appendix A, though some of the parameters have been updated since then as a consequence of our work.

\footnote{These bounties were published on November 1st 2021 at \url{https://www.zkhashbounties.info/}.}

**Our Contributions.** In this paper, we investigate the security offered by four different AO symmetric primitives, with a focus on attacks exploiting models of their round function as low degree polynomials. Indeed, the encoding of a primitive as a system of equations is not unique, and it is possible to find better ones than those considered by the authors in
their original cryptanalysis. This in turn leads to attacks that can violate in practice the security claims made for round-reduced versions.

For round-reduced Feistel–MiMC, we mount an attack based on univariate root-finding that showcases a significant issue with the initial security claims put forward by the Ethereum Foundation, and which allowed us to find a practical attack against an instance initially thought of as “hard”.

We also develop a general method to remove all the equations modelling the first 2 rounds of a Substitution-Permutation Network (SPN), provided that the first operation is the S-box layer rather than the linear layer, and that the S-box is a monomial—both of which usually hold. This has a significant impact on the complexity of solving the resulting system of equations, which allowed us to successfully apply a univariate root-finding algorithm to round-reduced Poseidon, and a multivariate one to round-reduced Rescue–Prime.

Finally, we show an efficient way to build algebraic attacks against the Ciminion encryption scheme. In particular, the asymptotic complexity of our algebraic attack is $\mathcal{O}(2^{4r\omega})$, while the designers expected the best attack to have complexity $\mathcal{O}(2^{6r\omega})$, with $r$ the number of rounds, and $\omega$ the exponent of matrix multiplication. Since they added extra rounds as a security margin, we cannot break practical parameters, but in theory our attack breaks parameters targeting large security levels (e.g. $s = 1024$). More importantly, it highlights the shortcomings of the initial security analysis of this algorithm.

**Paper Outline.** We start by introducing the necessary mathematical background and security definitions in Section 2. We then present general results on the resolution of polynomial systems in Section 3, both for multivariate and univariate techniques. We attack 3 out of 4 of the functions targeted by the Ethereum Foundation, namely Feistel–MiMC, Poseidon, and Rescue–Prime, in Section 4. Our cryptanalysis of the Ciminion encryption is in Section 5. We also present some detailed experimental results about the efficiency of our attacks in Section 6. Finally, Section 7 concludes the paper.

## 2 Preliminaries

### 2.1 Notations

In what follows, we use $\mathbb{F}_q$ to denote the finite field with $q$ elements. $q$ might be a prime number (Feistel–MiMC, Poseidon, Rescue–Prime, Ciminion) or a power of 2 (Feistel–MiMC, Ciminion). In the challenges designed by the Ethereum Foundation, $q = 2^{64} - 59$ is prime.

The set $\mathbb{F}_q^t$ is a vector space of $\mathbb{F}_q$ with canonical basis $\{e_0, ..., e_{t-1}\}$, where $e_i$ has only zero coordinates, except at position $i$ where it is equal to 1. We let $\rho_i : \mathbb{F}_q^t \to \mathbb{F}_q$ be the projection mapping $(x_0, ..., x_{t-1})$ to $x_i$.

The functions we consider operate over $\mathbb{F}_q^t$, where $t \in \{2, 3\}$. We then have that $F(x) = (F_0(x), ..., F_{t-1}(x))$, where each $F_i$ is of a coordinate of $F$ with $F_i = \rho_i \circ F$. Each $F_i$ can be uniquely written as

$$F_i(x_0, ..., x_{t-1}) = \sum_{K=(k_0, ..., k_{t-1})} c_i^{k_0, ..., k_{t-1}} \prod_{j=0}^{t-1} x_j^{k_j},$$

where $c_i^{k_0, ..., k_{t-1}}$ is a coefficient in $\mathbb{F}_q$. The **degree** of $F_i$ is

$$\deg(F_i) = \max_K \left\{ k_0 + ... + k_{t-1} \mid c_i^{k_0, ..., k_{t-1}} \neq 0 \right\},$$

and the degree of $F$ is the maximum degree of its coordinates.
2.2 Security Assessment

In this paper, we assess the security of several algorithms. Ciminion is an encryption scheme which uses a key of length $k$ to encrypt a given plaintext. In this case, the goal is simply to recover the key (or some subkeys) with a time complexity lower than $2^k$.

While that case was simple to define, most of this work deals with the analysis of cryptographic permutations, which are intended for use in a sponge construction [BDPVA07] to build secure hash functions. More precisely, we focus on Feistel–MiMC, Poseidon and Rescue–Prime. For such objects, the security requirement is a bit more subtle, but it is convincingly captured by the hardness of the following problem, nicknamed Constrained Input/Constrained Output (CICO).

Let $u < t$ be an integer, and let $Z_u$ be the vector space spanned by $\{e_0, ..., e_{t-u-1}\}$. In other words, $Z_u$ is the set of all the elements of $F_q^t$ such that their last $u$ coordinates are equal to 0 (or, equivalently, such that $\rho_i(x) = 0$ for all $t - 1 - u < i < t$).

Definition 1 (CICO Problem). Let $F : F_q^t \rightarrow F_q^t$ be a function, and let $u < t$ be an integer. The CICO problem consists in finding $x \in F_q^t$ such that

$$x \in Z_u \text{ and } F(x) \in Z_u.$$  

A brute-force approach would solve this problem with a time complexity of about $q^u$ calls to the permutation: the idea would simply be to try random values $x$ of $Z_u$ until one of them satisfies $F(x) \in Z_u$. The ability to solve this problem more efficiently than this brute-force search would potentially allow an attacker to find preimages for a given digest in a sponge mode, or could help with finding collisions, hence its relevance.

In what follows, we consider the case where $u = 1$. In this case, we use the simpler notation $Z = Z_1$.

The challenges described by the Ethereum Foundation are explicitly about solving the CICO problem for the permutations used by several AO hash functions, namely Feistel–MiMC, Poseidon, Rescue–Prime, and Reinforced Concrete. The details of the challenges are provided in Appendix A.

3 Systems of Polynomial Equations and their Resolution

Our attacks are based on modelling the cryptographic primitive using a system of polynomial equations, and then solving those using an off-the-shelf solver. The choice of the solving method has a significant impact on its performance. In general, in order to assess the security of a primitive against algebraic attacks, it is necessary to have some grip on the complexity of the various solving algorithms. In this section, we present some theoretical results on the resolution of univariate systems (Section 3.1), and then of multivariate systems (Section 3.2). Some experimental results on these attacks are presented in Section 6.

Overall, the univariate solving tends to be much more efficient. However, it cannot be applied to all algorithms as there are efficient methods to prevent its applicability, as was done by the designers of Rescue–Prime (see Section 4.4).

We assume that we can represent an attack against a cryptosystem with a well-defined system of $n$ variables $X_1 \ldots X_n$ in $F_q$, i.e composed of $n$ polynomial equations on $n$ variables:

$$\begin{align*}
P_1(X_1, \ldots, X_n) &= 0 \\
P_2(X_1, \ldots, X_n) &= 0 \\
&\vdots \\
P_n(X_1, \ldots, X_n) &= 0
\end{align*}$$
We assume that the system is non-trivial and that there exists a solution (in \( F_q^n \)) of this system that gives sufficient information to break the cryptosystem. This happens for example if a variable is a round key, if the variables describe a preimage under an algebraic hash function, or if we try to solve a CICO instance.

### 3.1 Solving Univariate Systems

In the univariate case, we have a system with a single equation and a single variable:

\[ P(X) = 0 \]

Solving the system is equivalent to finding the roots of the polynomial \( P \in F_q[X] \) in a finite field of characteristic \( q \); we denote the degree of \( P \) as \( d \).

We use fast arithmetic to multiply two polynomials of degree \( d \) with \( \mathcal{O}(d \log(d) \log(\log(d))) \) field operations using an FFT algorithm. Using the following method, finding the roots requires only \( \mathcal{O}(d \log^2(d) \log(\log(d))) \) field operations:

1. Compute \( Q = X^q - X \mod P \).
   Computing \( X^q \mod P \) requires \( \mathcal{O}(d \log(q) \log(d) \log(\log(d))) \) field operations using a double-and-add algorithm.

2. Compute \( R = \gcd(P, Q) \).
   \( R \) has the same roots as \( P \) in the field \( F_q \) since \( R = \gcd(P, X^q - X) \), but its degree is much lower (it is exactly the number of roots).
   This requires \( \mathcal{O}(d \log^2(d) \log(\log(d))) \) field operations.

3. Factor \( R \).
   In general, \( R \) is of degree one or two because \( P \) has few roots in the field, and this step is of negligible complexity.

In particular, finding the roots inside the field is significantly easier than factoring the polynomial (it is quasi-linear in the degree).

For practical instances, we use the NTL library [Sho], a C++ library for number theory, and the computation is feasible for a degree up to roughly \( 2^{32} \approx 3^{20} \) in a prime field \( F_p \) with \( p \approx 2^{64} \). We present some benchmarks in Section 6.1 for polynomials given by round-reduced version of Feistel–MiMC, of POSEIDON and for random polynomials.

### 3.2 Solving a Multivariate System

In general, we have a system with \( n \) polynomials in \( F_q[X_1, \ldots, X_n] \). A system is said to be regular if for all \( i, gP_i \in (P_1, \ldots, P_{i-1}) \Rightarrow g \in (P_1, \ldots, P_{i-1}) \). The Fröberg conjecture states that this is the typical behaviour for random polynomial systems [Frö85]. However, recent studies have shown that polynomial systems representing algebraic cryptographic algorithms are often not regular [DG10, DY13, Sau22]. A thorough analysis of specific non regular systems is very complex for the designers, therefore a common practice is to estimate the complexity of the Gröbner basis attack for an equivalent regular system, and to add extra rounds to account for the non-regularity of the system. If the system is well-defined (as much equations as variables) and if the system can not be reduced to strictly less than \( n \) equations (which holds for regular systems), then the number of solutions is finite in the algebraic closure of \( F_q^n \). Let us denote \( d_i \) the degree of the polynomial \( P_i \), \( d \) the degree of the ideal \( I = (P_1, \ldots, P_n) \), which is also the number of solutions in the algebraic closure of \( F_q^n \), and \( D_{\text{reg}} \) the degree of regularity of \( I \), as defined by Dubois et al. in [DG10]. We have

\[
D_{\text{reg}} \leq 1 + \Sigma_{i=1}^n (d_i - 1) \quad \text{and} \quad d \leq \Pi_{i=1}^n d_i .
\]
Moreover, the bounds are reached when the system is regular\(^2\).

The main technique to solve the multivariate polynomial systems is to compute a Gröbner basis [Buc76] of the ideal \( \mathcal{I} \). A Gröbner basis \( G \) of \( \mathcal{I} \), with respect to a total ordering on the set of monomials, is a particular generating set of \( \mathcal{I} \). Gröbner basis become very interesting under the lexicographic order since they take the form

\[
\{ G_1(x_1), G_{2,1}(x_1, x_2), \ldots, G_{2,k_2}(x_1, x_2), \ldots, G_{n,1}(x_1, \ldots, x_n), \ldots, G_{n,k_n}(x_1, \ldots, x_n) \}.
\]

Indeed, the solutions of this system are easy to recover iteratively by first computing the roots of \( G_1 \) in \( \mathbb{F}_q \), then substituting \( x_1 \) in polynomials \( G_{2,i} \), then computing the roots of \( G_{2,i} \), etc...

Computing a Gröbner basis from a polynomial system can be done with Buchberger’s algorithm [Buc76], or with more efficient algorithms such as F4 [Fau99] and F5 [Fau02] by Faugère. In practice, directly computing the Gröbner basis in lexicographic order is prohibitively expensive. Instead, Faugère et al. [FGLM93] proposed to first compute the Gröbner basis in another order, then to apply the FGLM algorithm to convert it into a Gröbner basis under the lexicographic order. To the best of our knowledge, the fastest algorithm to compute a Gröbner basis at the time of writing is Faugère’s F5 algorithm [Fau02] in the grevlex (for graded reverse lexicographic) monomial order. In the end, the overall approach to solving a multivariate system of equations follows the following steps.

1. Compute a grevlex order Gröbner basis with the F5 algorithm.
2. Convert it into a lexicographic order Gröbner basis using the FGLM algorithm.
3. Find the roots in \( \mathbb{F}_q^n \) of the Gröbner basis polynomials using univariate system resolution of subsection 3.1 and substitution.

**Complexity.** The complexity depends on the number of variables \( n \), the degree of the ideal \( d \) and the degree of regularity \( D_{\text{reg}} \). Below, we give estimates for the complexity of each step of the algorithm above.

1. Under the hypothesis that the polynomial system is regular, the complexity of the F5 algorithm is bounded by\(^3\)

\[
\mathcal{O} \left( \frac{n D_{\text{reg}} \times \left( n + D_{\text{reg}} - 1 \right)^2}{D_{\text{reg}}} \right) \omega,
\]

where \( 2 \leq \omega \leq 3 \) is the matrix multiplication exponent [BFS15, Proposition 1].

2. The original FGLM algorithm [FGLM93] has a complexity of \( \mathcal{O}(nd^3) \), but more recent variants achieve a better complexity with probabilistic methods. In particular, [FGHR14] reaches complexity \( \mathcal{O}(nd^ω) \), and [FM17] has complexity \( \mathcal{O}(\sqrt{n}d^{2+\frac{ω}{2}}) \).

3. Since all the roots are computed in \( \mathbb{F}_q \), we can use the fast univariate system solving of subsection 3.1 with substitution. Experimentally, in most cases we have a small number of roots and \( G_{1,1}(x_1) \) is of degree 1 after substitution of \( x_1 \ldots x_{i-1} \). This step is of complexity roughly \( \mathcal{O}(d \log^2(d)) \).

---

\(^2\)Note that the upper bound does not require the system to be regular.

\(^3\)In the security analysis of Ciminion, Rescue–Prime and POSEIDON, another correct upper bound \( \left( n + D_{\text{reg}} \right)^ω \) of [BFS04] is used. For small \( n \), the bound that we give is sharper, but not necessarily for large \( n \). In particular, our bound is better to estimate the complexity of the Ciminion attack of Section 5.
When the polynomials have the same degree $d_i = \delta$ and $n$ is small, the complexity of F5 is asymptotically smaller than the complexity of FGLM. With $d = \delta n$ and $D_{\text{reg}} = n \times \delta - n + 1$, we obtain

$$
\left( \frac{n + D_{\text{reg}} - 1}{D_{\text{reg}}} \right) = \left( \frac{n\delta}{n-1} \right) \leq \frac{(n\delta)^{n-1}}{(n-1)!} = \mathcal{O}(\delta^{n-1}) .
$$

On the other hand, when the polynomials have small degrees $d_i = \delta$ and $n$ is high, the complexity of F5 is asymptotically higher than the complexity of FGLM. Indeed, in this case we have $d = \delta n$ and $D_{\text{reg}} = n \times \delta - n + 1$, which implies

$$
\left( \frac{n + D_{\text{reg}} - 1}{D_{\text{reg}}} \right) = \left( \frac{n\delta}{n-1} \right) = n\delta, \quad \frac{n\delta - 1}{n-2}, \quad \ldots, \quad \frac{n\delta - n + 2}{1} \geq \delta^{n-2} .
$$

Note that these two properties are only valid for regular systems. In the case of non-regular systems, these upper bounds are not tight and the complexity of F5 may be smaller than expected, as shown in Section 6.2.

**Experimental results.** In practice, we use the Magma system, and the computation is feasible up to roughly a degree $d = 3^9$ in a prime field $\mathbb{F}_p$ with $p \approx 2^{64}$. Our experimental results for round-reduced versions of Rescue–Prime, Ciminion and random systems are summarized in Section 6.2. In practice, FGLM (step 2) is the bottleneck; Magma uses the F4 algorithm for step 1, which is almost as fast as F5, but its implementation of FGLM seems to have a complexity that is cubic in $d$.

## 4 CICO Cryptanalysis of Several Primitives

We now report our experimental results in solving the CICO problem for the challenges put forward by the Ethereum Foundation.

The complexities of some of our attacks are in line with designers’ claims. However, we have found that, in practice, the exact claimed security level for a given number of rounds is not always clear. Furthermore, breaking challenges in practice gives a more concrete understanding of the security of reduced versions since we focus on upper bounds rather than lower bounds. Besides, some designers’ security analysis rely on optimistic complexity assumptions (eg. ignoring log factors, or taking a small omega), so our attack may give more accurate security estimates.

### 4.1 Attacks Against Round-Reduced Feistel–MiMC

**Design description.** Feistel–MiMC is a Feistel network, based on the simple structure of MiMC, introduced by Albrecht et al. at Asiacrypt 2016 [AGR+16]. It operates on $\mathbb{F}_p^2 (t = 2)$ using a basic $r$-round Feistel structure with the $i$-th round function being

$$
x \mapsto (x + c_i)^\alpha \text{ (in this paper we take $\alpha = 3$, as fixed by the author of the Ethereum’s challenges).}
$$

![Figure 1: Round $i$ of Feistel–MiMC.](image-url)
**Attack description.** In order to build a polynomial system representing the CICO problem, we consider an input state \((P_0, Q_0) = (X, 0)\). Then we evaluate the round function iteratively, as polynomials in \(\mathbb{F}_p[X]\):

\[
\begin{align*}
P_0 &= X \\
P_i &= Q_{i-1} + (P_{i-1} + c_i)^3 \quad \text{for} \quad Q_0 = 0, \quad Q_i = P_{i-1}. 
\end{align*}
\]

The CICO problem becomes \(Q_r = 0\): we just have to find the roots of \(Q_r = P_{r-1}\).

In practice, we use SageMath to generate the polynomial, and we compute the roots either directly from SageMath, or with an external program using NTL. The corresponding code is given in the Supplementary Material.

**Complexity Analysis.** Since the round function has degree 3, we obtain a univariate polynomial \(P_{r-1}\) of degree \(3^{r-1}\) after \(r - 1\) rounds. We can estimate the complexity of finding the roots as:

\[
d \log(d) \left( \log(d) + \log(p) \right) \log(\log(d)) \approx 3^{r-1} \times (r - 1) \times 1.58 \times 64 \times \log_2(r - 1).
\]

We give explicit values for the proposed challenges in Table 1. Parameters have changed while we were working on it, so “original” (Table 1a) and “new parameters” (Table 1b) are two sets of parameters proposed by the Ethereum Foundation, the first ones being less secure than the latter ones.

We observe that the security claims from the Ethereum Foundation are close to \(3^{2r}\). This likely corresponds to an estimation of the complexity of a Gröbner base attack using \(r\) equations of degree 3 in \(r\) variables: the corresponding complexity would be \(3^{3r} \geq 3^{2r}\).

Besides, the original specification of Feistel–MiMC states that Lagrange interpolation attacks are expected to have a complexity of \(r \cdot 3^{2r-3}\), while GCDs attacks are expected to have a complexity of \(r^3 \cdot 3^{r/2-3}\). As the latter leaves more freedom to the attacker, it does not apply in our context. However, we have chosen to put both in Table 1 for a fairer comparison.

### 4.2 Bypassing SPN Steps

Let \(\pi = \pi_0 \circ \pi_1\) be a permutation of \(\mathbb{F}_p^t\), and \(\mathcal{Z}\) be the vector space spanned by \(\{e_0, \ldots, e_{t-2}\}\).

Suppose that there exists two vectors \(V\) and \(G\) in \(\mathbb{F}_p^t\) such that

\[
\pi_0^{-1}(XV + G) \in \mathcal{Z}
\]

for all \(X \in \mathbb{F}_p^t\). In this case, we write all the intermediate variables of \(\pi_1\) as polynomials in \(X\), starting from the state \(XV + G\), and evaluating round operations one by one as polynomials. Then we can find \(r\) such that \(\pi_1(rV + G) \in \mathcal{Z}\) by finding a root \(r\) of the polynomial corresponding to the last coordinate of the output. Finally, setting \(x = (x_0, x_1, \ldots, x_{t-1}) = \pi_0^{-1}(rV + G)\) will yield a solution to the CICO problem, while the solver has to handle a polynomial based on \(\pi_1\) rather than the full \(\pi\). This approach is summarized in Figure 2, and we used it against both Poseidon (see Section 4.3) and Rescue–Prime (see Section 4.4).

Let us describe this trick in more detail. First, for the sake of consistency, we will use steps when referring to the constant addition, the S-box, and the linear part. Then one round of Poseidon consists of one step, and one round of Rescue–Prime of two steps: one using \(S\) as S-box, the other using \(S^{-1}\).

We consider \(\pi_0\) to be two steps of an SPN construction without the final linear layer: addition of rounds constants, S-box layer \(S_1\), linear layer consisting of a multiplication by an MDS matrix, and S-Box layer \(S_2\). We require the S-boxes to be monomial functions, so
Table 1: Complexity of our attack against Feistel–MiMC, compared with the security claims given by the authors and by the challenges. Complexity figures in bold correspond to attacks that we have implemented in practice.

<table>
<thead>
<tr>
<th>$r$</th>
<th>Authors claims Lagrange</th>
<th>GCD</th>
<th>Ethereum claims $d$</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$2^{16}$</td>
<td>$2^5$</td>
<td>$2^{18}$</td>
<td>$3^5$</td>
</tr>
<tr>
<td>10</td>
<td>$2^{30}$</td>
<td>$2^9$</td>
<td>$2^{30}$</td>
<td>$3^9$</td>
</tr>
<tr>
<td>14</td>
<td>$2^{43}$</td>
<td>$2^{13}$</td>
<td>$2^{44}$</td>
<td>$3^{13}$</td>
</tr>
<tr>
<td>18</td>
<td>$2^{56}$</td>
<td>$2^{17}$</td>
<td>$2^{56}$</td>
<td>$3^{17}$</td>
</tr>
<tr>
<td>22</td>
<td>$2^{69}$</td>
<td>$2^{21}$</td>
<td>$2^{68}$</td>
<td>$3^{21}$</td>
</tr>
</tbody>
</table>

(a) Original parameters.

<table>
<thead>
<tr>
<th>$r$</th>
<th>Authors claims Lagrange</th>
<th>GCD</th>
<th>Ethereum claims $d$</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>$2^{69}$</td>
<td>$2^{21}$</td>
<td>$2^{36}$</td>
<td>$3^{21}$</td>
</tr>
<tr>
<td>25</td>
<td>$2^{79}$</td>
<td>$2^{24}$</td>
<td>$2^{40}$</td>
<td>$3^{24}$</td>
</tr>
<tr>
<td>30</td>
<td>$2^{95}$</td>
<td>$2^{28}$</td>
<td>$2^{48}$</td>
<td>$3^{29}$</td>
</tr>
<tr>
<td>35</td>
<td>$2^{111}$</td>
<td>$2^{33}$</td>
<td>$2^{56}$</td>
<td>$3^{34}$</td>
</tr>
<tr>
<td>40</td>
<td>$2^{127}$</td>
<td>$2^{37}$</td>
<td>$2^{64}$</td>
<td>$3^{39}$</td>
</tr>
</tbody>
</table>

(b) New parameters.

that $S(AX) = S(A)S(X)$. The question of whether the attack can be adapted to the case where the condition is not verified is an open problem.

We use $c^i_r$ to denote the $i$-th round constant used in step $r$. We let the linear layer $M$ be such that:

$$M^{-1} = \begin{bmatrix}
\alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{t-1,0} \\
\alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{t-1,1} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{0,t-1} & \alpha_{1,t-1} & \cdots & \alpha_{t-1,t-1}
\end{bmatrix}.$$  

**Case $t = 3$.** We start with the special case $t = 3$, and we denote the state after $\pi_0$ as $(X, Y, Z)$, with three variables. As seen in Figure 3, we have $\pi_0^{-1}(X, Y, Z) \in \mathcal{Z}$ if and only if

$$S_1(c^0_2) = \alpha_{0,2}S_2^{-1}(X) - c^0_3 + \alpha_{1,2}S_2^{-1}(Y) - c^1_3 + \alpha_{2,2}S_2^{-1}(Z) - c^2_3$$

$$= \alpha_{0,2}S_2^{-1}(X) + \alpha_{1,2}S_2^{-1}(Y) + \alpha_{2,2}S_2^{-1}(Z) - (\alpha_{0,2}c^0_0 + \alpha_{1,2}c^1_0 + \alpha_{2,2}c^2_0)$$

In order to simplify the equation, we fix $Z$ to a constant value $g$ with:

$$g = S_2(\alpha_{0,2}c^0_0 + \alpha_{1,2}c^1_0 + \alpha_{2,2}c^2_0 + S_1(c^0_2))$$. We obtain

$$\pi_0^{-1}(X, Y, g) \in \mathcal{Z} \iff \alpha_{0,2}(S_2^{-1}(X)) = -\alpha_{1,2}(S_2^{-1}(Y))$$

$$\iff S_2(\alpha_{0,2})X = S_2(-\alpha_{1,2})Y$$

Therefore, we obtain an affine space with $\pi_1(XV + G) \in \mathcal{Z}$ by choosing:

$$V = (1, S_2(\alpha_{0,2})/S_2(-\alpha_{1,2}), 0) \text{ and } G = (0, 0, g)$$.

**General case ($t \geq 3$).** In general, we take $V = (S_2(A_0), \ldots, S_2(A_{t-2}), 0)$ and $G = (0, \ldots, 0, g)$, such that we can consider an input state after the S-box layer of the second
step of the form \((S_2(A_0)X, \ldots, S_2(A_{t-2})X, g)\), and study the first two steps as shown in Figure 4.

Following Figure 4, the value \(S_1(c_{t-1}^0)\) must satisfy

\[
S_1(c_{t-1}^0) = \sum_{j=0}^{t-2} \alpha_{j,2}(A_jS_2^{-1}(X) - c_j^1) + \alpha_{t-1,2}(S_2^{-1}(g) - c_{t-1}^1)
\]

\[
= S_2^{-1}(X) \left( \sum_{j=0}^{t-2} \alpha_{j,2}A_j \right) + \alpha_{t-1,2}S_2^{-1}(g) - \sum_{j=0}^{t-1} \alpha_{j,2}c_j^1.
\]

It is the case provided for instance that:

\[
\begin{cases}
A_{t-2} = -\sum_{j=0}^{t-3} \frac{\alpha_{j+2}}{\alpha_{t-1,2}} A_j \\
g = S_2 \left( \frac{1}{\alpha_{t-1,2}} \sum_{j=0}^{t-1} \alpha_{j,2}c_j^1 + S_1(c_{t-1}^0) \right). \quad (1)
\end{cases}
\]

As a consequence, if we find a value \(X\) such that the image of \((S_2(A_0)X, \ldots, S_2(A_{t-2})X, g)\) through \(R - 2\) steps of the primitive is equal to \((*, \ldots, *, 0)\), then we will always be able to deduce an input \((x_0, x_1, \ldots, x_{t-2}, 0)\) for \(R\) steps of the primitive that is mapped to \(Z\).

### 4.3 Application to Round-Reduced Poseidon

**Design description.** Poseidon \([GKR^{+}21]\) is a family of hash functions, based on the HADES design strategy \([GLR^{+}20]\). The internal permutation is composed of \(r = RF + RP\) rounds of two different types: full rounds have \(t\) S-box functions, and partial rounds have only 1 S-box and \(t - 1\) identity functions. Each round function consists of adding the round constants\(^4\), applying partial or full S-box layers \(S\), and then multiplying the state by an MDS matrix \((M)\). The permutation starts with \(RF = RF/2\) full rounds, followed by \(RP\) partial rounds, and finally \(RF = RF/2\) full rounds.

The challenges from the Ethereum Foundation use \(t = 3\), the S-Box is \(x \mapsto x^3\) and \(RF = 8\) is fixed, while \(RP\) varies according to the security level required.

\(^4\)For the sake of consistency of the different hash functions presented in this paper, we will note the addition of constants: “AddC”.
**Figure 3:** Bypassing Two SPN Steps \((t = 3)\).

**Attack description.** A basic encoding of Poseidon into equations can be solved quickly for a small number of rounds. In fact, it was sufficient for us to be able to claim the first bounty offered by the Ethereum Foundation for this algorithm. However, targeting the next ones requires to use the technique described in Section 4.2. The idea is to decrease the degree and the complexity of the polynomial system by more carefully choosing its variables.

Let \(t = 3\), and \(S_1, S_2\) such that \(S_1(x) = S_2(x) = x^3\). Then, applying our trick for SPN rounds, we consider an input state after the S-box layer of the second round of the form \((\mathcal{A}_0 X, \mathcal{A}_1 X, g)\) (i.e. we use \(V = (\mathcal{A}_0^3, \mathcal{A}_1^3, 0)\) and \(G = (0, 0, g)\)). We obtain

\[
\begin{align*}
A_1 &= -\frac{\alpha_{0,2}}{\alpha_{1,2}} A_0 \\
g &= \left(\frac{1}{\alpha_{2,2}} (\alpha_{0,2} c_0^1 + \alpha_{1,2} c_1^1) + c_2^1 + (c_2^0)^3\right)^3.
\end{align*}
\]

As previously mentioned, if we find a value \(X\) such that the image of \((\mathcal{A}_0^3 X, \mathcal{A}_1^3 X, g)\) through \(R - 2\) rounds of Poseidon (and a linear layer) is equal to \((*, *, 0)\), then we will always be able to deduce an input \((x, y, 0)\) for \(R\)-round of Poseidon that is mapped to \(Z\).

Therefore, we evaluate the permutation as polynomials in \(\mathbb{F}_p[X]\) starting from the state \((A^3 X, B^3 X, g)\) with \(A, B, g\) satisfying System (2), and the CICO problem is equivalent to finding the root of the polynomial corresponding to the rightmost branch of the output.

In practice, we use **SageMath** to generate the polynomial, and we compute the roots either directly from **SageMath**, or with an external program using NTL. The corresponding code is given in the Supplementary Material.

**Complexity Analysis.** Poseidon has \(r = RF + RP\) rounds in total, but we skip the first two rounds using the trick. Therefore, we obtain a univariate polynomial of degree \(d = 3^{r-2}\), and we can estimate the complexity of finding the roots as:

\[
d \log(d \left( \log(d) + \log(p) \right) \log(\log(d))) \approx 3^{r-2} \times (r - 2) \times 1.58 \times 64 \times \log_2(r - 2).
\]

We give explicit values for the proposed challenges in Table 2, along with the corresponding security claims. For the challenges issued by the Ethereum foundation, the claim was that
84 Algebraic Attacks against Some Arithmetization-Oriented Primitives

\[ c_0 \oplus c_0 \oplus c_{t-2} \oplus c_{t-1} \]

\[ S_1 \ldots S_1 \]

\[ S_1(c_{t-1}) \]

\[ M \]

\[ A_0 S_2^{-1}(X) - c_0^1 \]

\[ A_{l-2} S_2^{-1}(X) - c_{l-2}^1 \]

\[ S_2^{-1}(g) - c_{l-1}^1 \]

\[ S_2(A_0)X \]

\[ S_2(A_{l-2})X \]

\[ g \]

**Figure 4:** Bypassing Two SPN Steps (general case).

**Figure 5:** Overview of the construction of Poseidon.

an attack would require at least \(2^{37+s}\) steps, where \(s\) is a “security” level specified in bits, and is equal to 8, 16, 34, 32 and 40 when \(RP\) is equal to 3, 8, 13, 19 and 24 respectively.

The original specification of Poseidon states that interpolation attacks are expected to have a complexity similar to the one of our attacks, namely about \(\alpha^{RP+RF} [GKR+21, Equation (3)]\). However, the challenges of the Ethereum Foundation\(^5\) appear to claim a higher security level.

### 4.4 Application to Round-Reduced Rescue–Prime

**Design description.** Rescue is a family of AO hash functions, that was first proposed as part of Marvellous designs [AAB+19]. Rescue has the particularity of using both a low degree S-box and its inverse. Indeed, each round of Rescue, consists of two steps: while the first one involves an S-box \(S\), an MDS matrix \(M\), and the addition of the round constants, the second one is quite similar but replaces \(S\) with its inverse \(S^{-1}\). The two steps of each round are described in Figure 6.

\(^5\)We observe that this claim is close to \(3^{3RF+RP}\), but it is unclear which attack it corresponds to.
Table 2: Complexity of our attack against Poseidon, compared with the security claims given by the authors and by the challenges, with RF = 8. Complexity figures in bold correspond to attacks that we have implemented in practice.

<table>
<thead>
<tr>
<th>RP</th>
<th>Authors claims</th>
<th>Ethereum claims</th>
<th>d</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$2^{17}$</td>
<td>$2^{45}$</td>
<td>3</td>
<td>$2^{26}$</td>
</tr>
<tr>
<td>8</td>
<td>$2^{25}$</td>
<td>$2^{53}$</td>
<td>3</td>
<td>$2^{35}$</td>
</tr>
<tr>
<td>13</td>
<td>$2^{33}$</td>
<td>$2^{61}$</td>
<td>3</td>
<td>$2^{44}$</td>
</tr>
<tr>
<td>19</td>
<td>$2^{42}$</td>
<td>$2^{69}$</td>
<td>3</td>
<td>$2^{54}$</td>
</tr>
<tr>
<td>24</td>
<td>$2^{50}$</td>
<td>$2^{77}$</td>
<td>3</td>
<td>$2^{62}$</td>
</tr>
</tbody>
</table>

For our study, we will use the specifications of Rescue–Prime [SAD20], which means in particular that in each round, we first apply $S$ and then $S^{-1}$ (rather than the contrary as described in the original paper [AAB+19]).

The challenges from the Ethereum Foundation use $t = 3$ or $t = 2$, and the S-boxes are $x \mapsto x^3$ and its inverse $x \mapsto x^{1/3}$.

For our study, we will use the specifications of Rescue–Prime [SAD20], which means in particular that in each round, we first apply $S$ and then $S^{-1}$ (rather than the contrary as described in the original paper [AAB+19]).

The challenges from the Ethereum Foundation use $t = 3$ or $t = 2$, and the S-boxes are $x \mapsto x^3$ and its inverse $x \mapsto x^{1/3}$.

**Figure 6:** Round $i$ of Rescue–Prime.

**Attack description.** Rescue–Prime cannot be efficiently written as a univariate polynomial system, because it uses both the S-boxes $x \mapsto x^3$ and $x \mapsto x^{1/3}$. Each S-box has a low univariate degree in one direction, but a high degree in the other direction. Therefore, we add intermediate variables so that each S-box can be described with a low-degree equation, and we build a multivariate system.

More precisely, let us consider Rescue–Prime with a $t$-element state ($t = 2$ or $t = 3$) and $N$ rounds. We use variables $(X_0, Y_0, \ldots)$ to represent the input and $(X_i, Y_i, \ldots)$ to represent the internal state after the $i$-th round ($t(N+1)$ variables in total). As shown in Figure 6, we can write $t$ equations linking the $t$ variables at the input and output of round $i$, using only the direct S-box $x \mapsto x^3$. Therefore, we have degree-3 equations:

$$\forall j \in \{1, \ldots, t\}, \quad P_{i,j}(X_i, Y_i, \ldots) - Q_{i,j}(X_{i+1}, Y_{i+1}, \ldots) = 0 .$$

If we add equations $X_0 = 0$ and $X_N = 0$, we obtain a system of polynomial equations representing the CICO problem. We observe that the input variables can be removed. Indeed, each $S(X_0), S(Y_0), \ldots$ can be written as degree-3 polynomial of $X_1, Y_1, \ldots$. Given that $S(X_0) = 0$, it follows that we can only keep the corresponding polynomial equal to 0, and then remove the input variables. We can also remove $X_N$ because it is fixed to zero, and we obtain a system of $t(N-1) + 1$ equations and $tN - 1$ variables.

With $t = 2$, we have the same number of equations and variables. However, with $t \geq 3$ we have more variables than equations, and we can use the trick of Section 4.2 to obtain a
smaller system corresponding to a subset of the solutions with one solution on average.

**Bypassing the First Round when \( t = 3 \).** Let us repeat the idea described in Section 4.3 and apply it to Rescue–Prime.

Let \( t = 3 \), and \( S_1, S_2 \) such that \( S_1(x) = x^3 \), and \( S_2(x) = S_1^{-1}(x) = x^{1/3} \). We consider an input state after the S-box layer of the second round of the form \((A_0^{1/3}X, A_1^{1/3}X, g)\) (i.e. we use \( V = (A_0^{1/3}, A_1^{1/3}, 0) \) and \( G = (0, 0, g) \)).

We first notice that we can switch the order of the multiplication by the MDS matrix and the addition of the constants. Let

\[
\begin{pmatrix}
C_0^0 \\
C_1^0 \\
C_2^0
\end{pmatrix} = M^{-1} \begin{pmatrix}
c_0^0 \\
c_1^0 \\
c_2^0
\end{pmatrix}.
\]

In particular, we have:

\[
C_2^0 = \alpha_{0,2}c_0^0 + \alpha_{1,2}c_1^0 + \alpha_{2,2}c_2^0.
\]

As a consequence, using the same notations as above, the value \( C_2^0 \), in Figure 7, must satisfy

\[
C_2^0 = \alpha_{0,2}A_0X^3 + \alpha_{1,2}A_1X^3 + \alpha_{2,2}g^3
= X^3(\alpha_{0,2}A_0 + \alpha_{1,2}A_1) + \alpha_{2,2}g^3.
\]

It is the case provided for instance when:

\[
\begin{align*}
A_1 &= -\frac{\alpha_{0,2}}{\alpha_{1,2}} A_0, \\
g &= \left( \frac{1}{\alpha_{2,2}} \left( \alpha_{0,2}c_0^0 + \alpha_{1,2}c_1^0 \right) + c_2^0 \right)^{1/3}.
\end{align*}
\]

Recalling that one round corresponds to two steps, it follows that, if we find a value \( X \) such that the image of \((A_0^{1/3}X, A_1^{1/3}X, g)\) through \( R - 1 \) rounds of Rescue–Prime (and a linear layer) is equal to \((*,*,0)\), then we will always be able to deduce an input \((x, y, 0)\) for \( R \)-round Rescue–Prime that is mapped to \( Z \).

![Figure 7: How to bypass the first round of Rescue–Prime.](image)

Then, for the remaining \( R - 1 \) rounds, Figure 6 shows how we generate the following polynomial equations to avoid the inverse S-box.

\[
\forall j \in \{0, 1, 2\}, \quad P_{i,j}(X_i, Y_i, Z_i) - Q_{i,j}(X_{i+1}, Y_{i+1}, Z_{i+1}) = 0.
\]
Finally, this results in the following system of polynomial equations:

\[
\begin{aligned}
\forall 1 \leq i \leq N - 1, \forall j \in \{0, 1, 2\}, \\
P_{i,j}(X_i, Y_i, Z_i) - Q_{i,j}(X_{i+1}, Y_{i+1}, Z_{i+1}) = 0,
\end{aligned}
\] (4)

where \(Z_N = 0\) and

\[
\begin{pmatrix}
X_1 \\
Y_1 \\
Z_1
\end{pmatrix} = M
\begin{pmatrix}
A_0^{1/3} X \\
A_1^{1/3} X \\
g
\end{pmatrix} +
\begin{pmatrix}
c_0^1 \\
c_1^1 \\
c_2^1
\end{pmatrix}.
\]

This system has \(t(N-1)\) variables and \(t(N-1)\) equations. As before, we used SageMath to generate our system of equation. However, we used Magma to find the solutions of the corresponding multivariate system. Again, our code is given in the Supplementary Material.

**Complexity Analysis.** With \(t = 3\) branches and \(N\) rounds, we obtain a system of \(3(N-1)\) degree-3 equations with the same number of variables. In our experiments, the system behaves like a generic system and has \(d = 3^{3(N-1)}\) solutions in the algebraic closure of the field. Therefore, the complexity of solving the system is approximately:

\[
d^\omega \leq d^3 = 3^{9(N-1)}.
\]

With \(t = 2\) branches and \(N\) rounds, we obtain a system of \(2N-1\) degree-3 equations with the same number of variables. Therefore, \(d = 3^{2N-1}\) and the complexity of solving the system is approximately:

\[
d^\omega \leq d^3 = 3^{6N-3}.
\]

Besides, the original paper of Rescue–Prime states that Gröbner basis attacks are expected to have the following complexity

\[
\left(\frac{(0.5(\alpha - 1) + 1)t(N-1) + 3}{t(N-1) + 1}\right)^2 [SAD20, Section 2.5].
\]

We give explicit values for the proposed challenges in Table 3.

**Table 3:** Complexity of our attack against Rescue–Prime, compared with the security claims given by the authors and by the challenges. Complexity figures in bold correspond to attacks that we have implemented in practice.

<table>
<thead>
<tr>
<th>(N)</th>
<th>(t)</th>
<th>Authors claims</th>
<th>Ethereum claims</th>
<th>(d)</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>(2^{36})</td>
<td>(2^{37.5})</td>
<td>(3^9)</td>
<td>(2^{43})</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>(2^{40})</td>
<td>(2^{37.5})</td>
<td>(3^{11})</td>
<td>(2^{53})</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>(2^{48})</td>
<td>(2^{43.5})</td>
<td>(3^{13})</td>
<td>(2^{62})</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>(2^{48})</td>
<td>(2^{45})</td>
<td>(3^{12})</td>
<td>(2^{57})</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>(2^{56})</td>
<td>(2^{49.5})</td>
<td>(3^{15})</td>
<td>(2^{72})</td>
</tr>
</tbody>
</table>

**5 Attacks Against Ciminion**

**Design description.** Ciminion is a symmetric encryption scheme from Dobroaunig *et al.* published at Eurocrypt 2021 [DGGK21] that aims to minimize the number of multiplications in large finite fields. Unlike Feistel–MiMC, POSEIDON and Rescue–Prime, Ciminion does not use a power map S-box (such as \(x \rightarrow x^3\)) and the non-linear diffusion instead comes from the use of Toffoli gates \((a, b, c) \mapsto (a, b, c + ab)\). In addition, Ciminion uses a
light linear layer instead of an MDS matrix. Ciminion’s encryption scheme is presented in Figure 8. For the sake of consistency with previous figures, we show it on $\mathbb{F}_p$ rather than $\mathbb{F}_{2^n}$ as presented in the original paper. $pc$ and $pE$ are both permutations based on the same round function, presented in Figure 10. For a security level of $s$, $pc$ possesses $s + 6$ rounds and $pE \max\{\lceil \frac{s+37}{12} \rceil, 6\}$ rounds. $rol$ is a non-linear rolling function described in Figure 9.

Dobraunig et al. performed a thorough security analysis of Ciminion, exploring all cryptanalysis techniques, such as linear cryptanalysis, differential cryptanalysis, higher-order differentials, interpolation and Gröbner basis attacks. For the latter however, they studied a modified Ciminion that they conjectured to be weaker than the real Ciminion. In the modified Ciminion, they came up with a system of 6 equations of degrees $\{2^r-1, 2^r, 2^{r+1}, 2^{r+1}, 2^{r+2}\}$ over 6 variables, where $r$ is the number of rounds of $pE$.

The value of $r$ was chosen so that this attack has complexity at least $2^s$. More precisely,
the authors estimated the complexity of the F5 algorithm with parameters
\[ n = 6 \quad \text{and} \quad D_{\text{reg}} = 21 \cdot 2^r - 5 \approx 2^{r+3.4} . \]

Following [BFS04], they estimated the complexity\(^6\) as
\[
\left( \frac{n + D_{\text{reg}}}{D_{\text{reg}}} \right)^\omega \leq \left( \frac{(D_{\text{reg}} + n)^n}{n!} \right)^\omega \approx 2^{(6r+10.9)\omega} .
\]

The designers took \( \omega = 2 \) as a lower bound, obtaining a minimum number of rounds
\[ r \geq \lceil \frac{n - 21.9}{12} \rceil , \]
and added 5 rounds as a security margin.

**A new polynomial system.** Instead of looking at a system of equations resulting from a
presumed weaker scheme, we study the real scheme and propose a better way to set up a
system of equations.

For a given nonce \( \aleph \), we consider the first two output blocks. We denote \( \alpha_i = C_i - P_i \)
and \( \alpha'_i = C'_i - P'_i \), for \( i = 1 \ldots 4 \), and introduce two variables \( X, Y \in \mathbb{F}_q \) for the missing
output words (not given as part of the ciphertext) after the first and second permutations
\( p_E \) (see Figure 11). The output of the first permutation \( p_E \) is \( (\alpha_1, \alpha_2, X) \), therefore, we
can write the input as polynomials in \( X \):
\[
\left( Q_{0}^{\alpha_1,\alpha_2}(X), Q_{1}^{\alpha_1,\alpha_2}(X), Q_{2}^{\alpha_1,\alpha_2}(X) \right) = p_{E}^{-1}(\alpha_1, \alpha_2, X)
\]

Similarly, the output of the second permutation \( p_E \) is \( (\alpha_3, \alpha_4, Y) \), and we can write the
corresponding input as polynomials in \( Y \):
\[
\left( Q_{0}^{\alpha_3,\alpha_4}(Y), Q_{1}^{\alpha_3,\alpha_4}(Y), Q_{2}^{\alpha_3,\alpha_4}(Y) \right) = p_{E}^{-1}(\alpha_3, \alpha_4, Y)
\]

---

\(^6\)This estimation is actually an upper bound, and a sharper upper bound of \( O \left( nD_{\text{reg}} \times \left( \frac{n + D_{\text{reg}} - 1}{D_{\text{reg}}} \right)^\omega \right) \) is given in [BFS15, Proposition 1] as mentioned in Section 3.2.
Then, we write equations linking the input of the first two \( p_E \) through the rol function:

\[
\begin{align*}
Q_0^{\alpha_1, \alpha_2}(X) &= Q_1^{\alpha_1, \alpha_4}(Y) - K_4 \\
Q_1^{\alpha_1, \alpha_2}(X) &= Q_2^{\alpha_2, \alpha_4}(Y) - K_3 \\
Q_2^{\alpha_1, \alpha_2}(X) &= Q_0^{\alpha_1, \alpha_4}(Y) - Q_1^{\alpha_3, \alpha_4}(Y) \circ Q_2^{\alpha_3, \alpha_4}(Y)
\end{align*}
\]

Finally, taking two nonces \( N, N' \), we eliminate the keys \( K_3, K_4 \) and obtain a system of four equations in the four variables \((X, Y, X', Y')\), using two blocks of ciphertexts from each nonce:

\[
\begin{align*}
Q_0^{\alpha_1, \alpha_2}(X) - Q_0^{\alpha_1, \alpha_2}(X') &= Q_1^{\alpha_3, \alpha_4}(Y) - Q_1^{\alpha_3, \alpha_4}(Y') \\
Q_1^{\alpha_1, \alpha_2}(X) - Q_1^{\alpha_1, \alpha_2}(X') &= Q_2^{\alpha_3, \alpha_4}(Y) - Q_2^{\alpha_3, \alpha_4}(Y') \\
Q_2^{\alpha_1, \alpha_2}(X) &= Q_0^{\alpha_3, \alpha_4}(Y) - Q_1^{\alpha_3, \alpha_4}(Y)Q_2^{\alpha_3, \alpha_4}(Y) \\
Q_2^{\alpha_1, \alpha_2}(X') &= Q_0^{\alpha_3, \alpha_4}(Y') - Q_1^{\alpha_3, \alpha_4}(Y')Q_2^{\alpha_3, \alpha_4}(Y').
\end{align*}
\]

Solving this system allows to recover the full internal state, and to deduce the keys \( K_1, K_2, K_3, K_4 \). In order to solve the system, we use the approach explained in Section 3.2.

**Solving complexity.** Let us denote \( \mathcal{P}_i \), the polynomial corresponding to the \( i \)-th row of the system, such that we have for all \( i = 1, \ldots, 4 \), \( \mathcal{P}_1(X, X', Y, Y') = 0 \). We notice that \( \mathcal{P}_i \) is a sum of univariate polynomials: \( \mathcal{P}_i(X, X', Y, Y') = \mathcal{P}_1^X(X) + \mathcal{P}_1^X(X') + \mathcal{P}_1^Y(Y) + \mathcal{P}_1^Y(Y') \). Then, it is sufficient to determine max, \( \deg(\mathcal{P}_i) \) for each \( \mathcal{P}_i \). Given that \( \deg(Q_0) = 2^{r-1} \), \( \deg(Q_1) = 2^{r-1} \), and \( \deg(Q_2) = 2^r \), the degree of the \( \mathcal{P}_i \) polynomials in \( X, X', Y \) and \( Y' \) are:

<table>
<thead>
<tr>
<th>( \mathcal{P}_i )</th>
<th>( X )</th>
<th>( X' )</th>
<th>( Y )</th>
<th>( Y' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{P}_1 )</td>
<td>( 2^{r-1} )</td>
<td>( 2^{r-1} )</td>
<td>( 2^{r-1} )</td>
<td>( 2^{r-1} )</td>
</tr>
<tr>
<td>( \mathcal{P}_2 )</td>
<td>( 2^{r-1} )</td>
<td>( 2^{r-1} )</td>
<td>( 2^r )</td>
<td>( 2^r )</td>
</tr>
<tr>
<td>( \mathcal{P}_3 )</td>
<td>( 2^r )</td>
<td>( 0 )</td>
<td>( 3 \cdot 2^{r-1} )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( \mathcal{P}_4 )</td>
<td>( 0 )</td>
<td>( 2^r )</td>
<td>( 0 )</td>
<td>( 3 \cdot 2^{r-1} )</td>
</tr>
</tbody>
</table>

In particular, system (5) has 2 equations of degree \( 3 \cdot 2^{r-1} \), 1 of degree \( 2^r \), and 1 of degree \( 2^{r-1} \). Therefore, we have the following parameters:

\[
n = 4 \quad D_{\text{reg}} \leq 1 + \sum_{i=1}^{r} (d_i - 1) \approx 2^{r+2.2} \quad d \leq \Pi_{i=1}^{n} d_i \approx 2^{4r+0.2}.
\]

We can deduce upper bounds on the cost of the steps required to solve the system.

Computing a Gröbner basis with respect to the \textit{grevlex} order using Faugère’s F5 algorithm has asymptotic complexity

\[
n D_{\text{reg}} \times \left( \frac{n + D_{\text{reg}} - 1}{D_{\text{reg}}} \right)^{\omega} = 4 \times 2^{r+2.2} \times \left( \frac{2^{r+2.2} + 3}{2^r + 2.2} \right)^{\omega} \leq 2^{r+4.2} \left( \frac{2^{r+2.2} + 3}{3!} \right)^{\omega} \approx 2^{r+4.2 + (3r+4)\omega}.
\]

On the other hand, performing the change of order with a fast variant of FGLM has asymptotic complexity

\[
d^{\omega} \approx 2^{(4r+0.2)\omega}.
\]

FGLM is therefore the bottleneck.
From an attacker point of view, we assume that linear algebra is implemented with Strassen's algorithm, resulting in $\omega = 2^{0.807}$ (asymptotically, the best algorithm known has $\omega < 2^{0.373}$, but only for implausibly large sizes). Taking the designer's recommended number of rounds $r = \lceil s + 37 \rceil$, this attack is slightly faster than $2^s$ for large values of $s$, with a time complexity

$$2^{(4r + 0.2)\omega} = 2^{(4\lceil s + 37 \rceil + 0.2)\omega} \approx 2^{4s + 35.2} \approx 2^{0.94s + 35.2}.$$ 

For practical values of $s$, the attack is not faster than $2^s$, but it shows that the design has much less security margin than anticipated by the designers. In particular, if we take an optimistic value $\omega = 2$ as in the security analysis of the designers, we obtain an attack with complexity roughly $2^{11.4}$ for the 128-bit security version recommended by the designers with 14 rounds.

6 Experimental Results

In order to better understand the behaviour of the root-finding tools we relied on in our attacks, we performed additional benchmarks on top of our attacks against the Ethereum challenges. We treat the cases of univariate and multivariate equations separately.

6.1 Univariate Solving

For root-finding of univariate polynomials, we investigated the FLINT [Har11] and NTL C libraries. Both support operations related to big polynomials in finite fields, but NTL was considerably faster for different sizes of toy polynomials, therefore we chose to benchmark only NTL. In order to work with high degree polynomials with NTL, we need to apply a small patch to the library source files to increase the value of NTL_FFTMaxRoot.

Table 4 presents our experimental results, with $p = 18446744073709551557 \approx 2^{64}$. Given a degree $d = 3^k$, we generate the polynomial modeling the CICO problem for Poseidon with $(RP, RF) = (k - 6, 8)$ and for Feistel–MiMC with $k + 1$ rounds, and random polynomials (each of the $d + 1$ coefficients is taken randomly in $F_p$). For each instance and degree, we launched $\min(32, 2^{18-k})$ jobs, with varying random polynomials. For all instances, the standard deviation of the memory consumption is negligible (in average $10^{-5}$ times the average value), and the standard deviation of time stays under 3 percent of the average value.

The data is represented in Figure 12, and we performed a linear regression of the time and memory usage of random polynomials root finding. We notice that the structure of the polynomials of Feistel–MiMC and Poseidon does not offer a significant speed up to the root finding compared to random polynomials.

Because the theoretical complexity is quasi-linear, the linear regression should be treated cautiously. In addition, the benchmarks apply only on 1 core and do not account for parallelization. We expect to speed up the univariate root finding with NTL parallelization (which, officially, is supported), but some tests showed that NTL CPU usage does not exceed 300%, even with more than 3 threads.

6.2 Multivariate Solving

For benchmarks of multivariate solving, we chose to use Magma [BCP97]. We compare the resources needed for the resolution of the Rescue–Prime and Ciminion polynomial systems to the resources needed for random equivalent systems. It should be noted that Magma implements the F4 algorithm [FGHR14] to find the grevlex Gröbner basis, and the FGLM algorithm [FGLM93] in cubic complexity for the change of ordering. Also, there seems to
Table 4: Benchmarks of univariate root finding with NTL (using 1 core of an Intel Xeon E7-4860), for 
POSEIDON, Feistel–MiMC and random polynomials of several degrees, with 
$p = 18446744073709551557 \approx 2^{64}$. Times are given in seconds and memory usage in 
MegaBytes.

<table>
<thead>
<tr>
<th>System</th>
<th>Degree</th>
<th>$3^{11}$</th>
<th>$3^{12}$</th>
<th>$3^{13}$</th>
<th>$3^{14}$</th>
<th>$3^{15}$</th>
<th>$3^{16}$</th>
<th>$3^{17}$</th>
<th>$3^{18}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feistel–MiMC</td>
<td>$X^p \mod P$ time</td>
<td>13 54 148 535 1,426 5,119 14,243 46,256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCD time</td>
<td>7 23 78 261 889 2,970 9,687 36,451</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td>20 77 226 796 2,315 8,089 23,930 82,707</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>104 293 822 2,431 6,967 20,696 69,227 2.07 · 10^5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSEIDON</td>
<td>$X^p \mod P$ time</td>
<td>13 54 148 534 1,454 5,083 14,241 47,963</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCD time</td>
<td>8 23 78 262 893 2,964 9,699 38,541</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td>21 77 226 796 2,347 8,047 23,940 86,504</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>104 293 838 2,431 6,968 20,696 69,538 2.07 · 10^5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>$X^p \mod P$ time</td>
<td>14 56 152 547 1,433 5,117 14,406 47,964</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCD time</td>
<td>7 23 80 269 903 2,976 9,790 38,693</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td>21 79 232 816 2,336 8,093 24,196 86,657</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>102 293 822 2,431 6,935 20,696 60,538 2.07 · 10^5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 12: Benchmarks of univariate root finding with NTL (using 1 core of an Intel Xeon 
E7-4860).
be a fixed memory overhead of 32 MB when using Magma for Gröbner basis, therefore we did not take into account low-memory points in the linear regressions.

Rescue–Prime and Rescue–Prime-like systems Table 5 and Figure 13 present the results for the resolution of a $k$-round Rescue–Prime instance, $k = 3, 4$, with $m = 3$ and $p = 18446744073709551557 \approx 2^{64}$. These are compared to random equivalent systems of $n$ equations of degree 3 on $n$ variables, generated randomly by affecting a random coefficient of $F_p$ to each possible monomial of degree $\leq 3$. In order to give a better insight on the evolution of the resources consumption, we chose to additionally benchmark random systems with $n = 5, 7, 8$, which do not correspond to any version of Rescue–Prime.

For systems of $n$ equations, we launched $\min(2^{2(9-n)}, 64)$ jobs. The standard deviation of time and memory consumption never exceeds 2% of the average value. The program was cut after 7 days for random systems with $n = 9$. The F4 step finished, but the FGLM step did not finish (after approximately 44 hours). The linear regression of Rescue system FGLM time might be biased compared to Random FGLM time, because only Rescue possesses a data point with $n = 9$, which demands heavy resources in memory, potentially causing some overhead.

The results highlight several properties:

- We observe that the theoretical maximal ideal degree is reached, for all systems: $3^n$ for $n$ equations of degree 3.

- The F4 run time varies considerably between the Rescue–Prime system and a random system. For 4-round Rescue–Prime ($n = 9$), there is almost a factor 50 between the run time of F4 on the Rescue–Prime system and on a random system.

- F4 dominates in time for random system, but FGLM heavily dominates for Rescue–Prime systems.

- The case with 6 equations is the only point of comparison between the Rescue–Prime system and the random system, but on this data point, FGLM is faster on Rescue–Prime than on a random system, despite having the same ideal degree (729).

- The memory consumption seems to essentially follow the same linear regression for both Rescue–Prime systems and Random systems.

Ciminion and Ciminion-like systems For $r$ rounds of Ciminion, in Section 5, we present a modelization of the cryptosystem with 4 equations on 4 variables, of degrees respectively $2^{r-1}, 2^r, 3 \cdot 2^{r-1},$ and $3 \cdot 2^{r-1}$. For the sake of simplicity, we kept the same prime number $p = 18446744073709551557 = 2^{64} - 59$, although it is less than $2^{64}$ (Ciminion normally requires a prime $p > 2^{64}$). We chose to add the concept of half rounds to increase the number of data points: $r + 0.5$ rounds of Ciminion is Ciminion where the first branch $p_E$ has undergone $r + 1$ rounds while the second branch $p_E$ has only been through $r$ rounds. With the same technique as in Section 5, we can represent this instance of Ciminion with a system of 4 equations on 4 variables, of degrees $2^{r-1}, 2^{r-1}, 2^{r-1},$ and $2^{r-1}$. We compare the Ciminion systems to random systems of 4 equations on 4 variables with the same degrees, where a random coefficient of $F_p$ is assigned to every possible monomial in each equation. The 4-round Ciminion was cut off due to memory insufficiency ($\geq 192$ GB).

Table 6 and Figure 14 present the results. We did not take into account the memory points with $r = 2$ in the linear regression, because it seems to be a fixed overhead when solving Gröbner bases with Magma (regardless of their sizes).

The results allow us to make the following observations.
Table 5: Benchmarks of multivariate root finding with Magma using 1 CPU core of an Intel Xeon Gold 5218, for Rescue and Rescue-like systems. Times are given in seconds and memory usage in MegaBytes.

<table>
<thead>
<tr>
<th>System</th>
<th>Number of equations</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rescue–Prime</td>
<td>Rounds</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F4 time</td>
<td>1.41</td>
<td></td>
<td></td>
<td>8,500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FGLM time</td>
<td>7.77</td>
<td></td>
<td></td>
<td>2.5 \cdot 10^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>112</td>
<td></td>
<td></td>
<td>58,675</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ideal degree</td>
<td>729</td>
<td></td>
<td></td>
<td>19,683</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>F4 time</td>
<td>0.25</td>
<td>8.23</td>
<td>299</td>
<td>11,120</td>
<td>4.46 \cdot 10^5</td>
</tr>
<tr>
<td></td>
<td>FGLM time</td>
<td>0.58</td>
<td>11.15</td>
<td>263</td>
<td>6,490</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>32</td>
<td>134</td>
<td>936</td>
<td>7,945</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ideal degree</td>
<td>243</td>
<td>729</td>
<td>2,187</td>
<td>6,561</td>
<td>19,683</td>
</tr>
</tbody>
</table>

Figure 13: Benchmarks of multivariate root finding with Magma using 1 CPU core of an Intel Xeon Gold 5218, for Rescue–Prime and Rescue–Prime-like systems (n equations of degree 3), with p = 18446744073709551557 and m = 3. 3-round and 4-round Rescue-prime respectively correspond to n = 6 and n = 9.
• The Ciminion system of equation does not reach the maximal ideal degree. We did not succeed to find a simple reduction of the system to explain this fact. This is surprising and not accounted for in the security analysis of the designer of Ciminion.

• The FGLM step heavily dominates the time complexity for Ciminion-like systems. For half rounds, the 4 equations all have the same degree, and this confirms the property showed in Section 3.2.

• We observe a factor 20 between the time complexity of the F4 step of the Ciminion system and random equivalent systems.

• We observe a factor at least 2.5 between the time complexity of the FGLM step of the Ciminion system and random equivalent systems. This is partially due to the suboptimal ideal degree of the Ciminion system.

• Extrapolating the results to $s = 64$ and $r = 9$ (since $q \approx 2^{64}$ it does not make sense to consider larger security levels), the expected time complexity of the FGLM step is $9 \cdot 10^{-8} \cdot 2^{10.529} = 2^{90.8}$ seconds (and not operations), which gives a comfortable security margin: our modelization does not break Ciminion.

For Ciminion and for Rescue–Prime systems, the FGLM step is predominant. However, the implementation of FGLM in Magma is in cubic complexity on the ideal degree. With a more performant implementation of FGLM [FGHR14, FM17], we should reduce the time complexity for both systems.

7 Conclusion

Our results show that properly choosing the parameters for arithmetization-oriented primitives is still non-trivial. In particular, some of the parameters chosen for the Ethereum Foundation challenge were apparently erroneous.

More specifically, we suggest a few lessons that can be learnt from our findings:

Encodings. There are many different ways to set up a system of equations corresponding to an attack against a primitive, but different encoding can result in different attack complexities. Designers should consider as many variants as possible when evaluating the complexity of algebraic attacks.

Univariate solving. When a primitive can be modelled as a univariate polynomial, this is usually the most efficient way to mount an algebraic attack. Indeed, finding the roots (in the field) of a random univariate polynomial has a complexity that is quasi-linear in the degree. Therefore, the number of rounds should be defined by the degree of the univariate polynomial, rather than by the complexity of computing a Gröbner basis.

First round. If the first round starts with a layer of S-boxes, this layer can be ignored when attacking the CICO problem. It is then better to start (and end) with a linear diffusion layer.

Second round. When $t \geq 3$, the linearization trick from Section 4.2 allows to skip one additional round when attacking the CICO problem with $u = 1$.

Acknowledgments

We thank the ToSC reviewers for their detailed comments which helped improve the clarity of this paper, and Lorenzo Grassi for shepherding it. We also thank Jules Baudrin and
Table 6: Benchmarks of multivariate root finding with Magma using 1 CPU core of an Intel Xeon Gold 5218, for Ciminion and Ciminion-like systems. Times are given in seconds and memory usage in MegaBytes, for systems of \( n \) equations of degree 3-round and 4-round Rescue-prime respectively correspond to \( n = 6 \) and \( n = 9 \).

<table>
<thead>
<tr>
<th>System</th>
<th>Rounds</th>
<th>Expected Ideal degree</th>
<th>288</th>
<th>1024</th>
<th>4608</th>
<th>16384</th>
<th>73728</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciminion</td>
<td>F4 Time</td>
<td>( 2 \cdot 10^{-2} )</td>
<td>0.23</td>
<td>6.7</td>
<td>209.1</td>
<td>13,848</td>
<td>5,624</td>
</tr>
<tr>
<td></td>
<td>FGLM Time</td>
<td>32</td>
<td>125.66</td>
<td>1,279</td>
<td>19,040</td>
<td>43,840</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ideal degree</td>
<td>170</td>
<td>680</td>
<td>2,736</td>
<td>10,944</td>
<td>43,840</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>F4 time</td>
<td>0.1</td>
<td>2.36</td>
<td>92.9</td>
<td>3,030</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FGLM time</td>
<td>0.74</td>
<td>18.96</td>
<td>1,011</td>
<td>32,069</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>32</td>
<td>226</td>
<td>3,480</td>
<td>47,444</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ideal degree</td>
<td>288</td>
<td>1,024</td>
<td>4,608</td>
<td>16,384</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 14: Benchmarks of multivariate root finding with Magma using 1 CPU core of an Intel Xeon Gold 5218, for Ciminion and Ciminion-like systems (4 equations of high degree).
Clara Pernot for proof reading a first draft of this manuscript, and Magali Bardet and Pierre Briaud for helpful discussions about solving multivariate systems.

Finally, we thank the Ethereum foundation for putting forward the challenges that kickstarted the research presented in this paper.

References


A Challenge Parameters

The challenges set out by the Ethereum foundation had the following parameters and claims\(^7\):

**Rescue–Prime [SAD20]:**

We expect that a variant with \(s\) bits of security to withstand attacks of complexity up to \(2^{A(1.5s)}\) time (function calls) and memory.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
<th>Security Level (bits)</th>
<th>Bounty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>(N=4, m=3)</td>
<td>25</td>
<td>$2,000</td>
</tr>
<tr>
<td>Easy</td>
<td>(N=6, m=2)</td>
<td>25</td>
<td>$4,000</td>
</tr>
<tr>
<td>Medium</td>
<td>(N=7, m=2)</td>
<td>29</td>
<td>$6,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(N=5, m=3)</td>
<td>30</td>
<td>$12,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(N=8, m=2)</td>
<td>33</td>
<td>$26,000</td>
</tr>
</tbody>
</table>

**Feistel-MIMC [AGR\(^+\)16]:**

We expect that a variant with \(s\) bits of security to withstand attacks of complexity up to \(2^{A(2s)}\) time (function calls) and memory.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
<th>Security Level (bits)</th>
<th>Bounty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>(r=6)</td>
<td>9</td>
<td>$2,000</td>
</tr>
<tr>
<td>Easy</td>
<td>(r=10)</td>
<td>15</td>
<td>$4,000</td>
</tr>
<tr>
<td>Medium</td>
<td>(r=14)</td>
<td>22</td>
<td>$6,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(r=18)</td>
<td>28</td>
<td>$12,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(r=22)</td>
<td>34</td>
<td>$26,000</td>
</tr>
</tbody>
</table>

**Poseidon [GKR\(^+\)21]:**

We expect that a variant with \(s\) bits of security to withstand attacks of complexity up to \(2^{A(s+37)}\) time (function calls) and memory.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
<th>Security Level (bits)</th>
<th>Bounty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>(RP=3)</td>
<td>8</td>
<td>$2,000</td>
</tr>
<tr>
<td>Easy</td>
<td>(RP=8)</td>
<td>16</td>
<td>$4,000</td>
</tr>
<tr>
<td>Medium</td>
<td>(RP=13)</td>
<td>24</td>
<td>$6,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(RP=19)</td>
<td>32</td>
<td>$12,000</td>
</tr>
<tr>
<td>Hard</td>
<td>(RP=24)</td>
<td>40</td>
<td>$26,000</td>
</tr>
</tbody>
</table>

\(^7\) An archived version of the challenge page is available at [https://web.archive.org/web/20211101211629/https://www.zkhashbounties.info/](https://web.archive.org/web/20211101211629/https://www.zkhashbounties.info/) but the images are missing.
Reinforced Concrete [GKL+21]:

We expect that a variant with $s$ bits of security to withstand attacks of complexity up to $2^s(2s)$ time (function calls) and memory.

| Decomposition and alpha/beta values |

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
<th>Security Level (bits)</th>
<th>Bounty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>$p=28177497670597$</td>
<td>24</td>
<td>$4,000$</td>
</tr>
<tr>
<td>Medium</td>
<td>$p=7205759403926839$</td>
<td>28</td>
<td>$6,000$</td>
</tr>
<tr>
<td>Hard</td>
<td>$p=184667440739555557$</td>
<td>32</td>
<td>$12,000$</td>
</tr>
</tbody>
</table>

On November 23rd (after we had sent solution to the first three Feistel–MiMC challenges), the Feistel–MiMC challenges were modified as follows:

Feistel-MiMC:

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
<th>Security Level (bits)</th>
<th>Bounty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>$r=22$</td>
<td>18</td>
<td>$4,000$</td>
</tr>
<tr>
<td>Easy</td>
<td>$r=25$</td>
<td>20</td>
<td>$6,000$</td>
</tr>
<tr>
<td>Medium</td>
<td>$r=30$</td>
<td>24</td>
<td>$6,000$</td>
</tr>
<tr>
<td>Hard</td>
<td>$r=35$</td>
<td>28</td>
<td>$12,000$</td>
</tr>
<tr>
<td>Hard</td>
<td>$r=40$</td>
<td>32</td>
<td>$26,000$</td>
</tr>
</tbody>
</table>