Practical Algebraic Attacks against some Arithmetization-oriented Hash Functions

Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, Léo Perrin

Practical Algebraic Attacks against some Arithmetization-oriented Hash Functions

Augustin Bariant[1], Clémence Bouvier[2,1], Gaëtan Leurent[1], Léo Perrin[1]

[1] Inria, France

[2] Sorbonne University, France

Abstract. Several challenges have been announced on arithmetization-oriented hash functions, with bounties funded by the Ethereum Foundation. In this note, we report on our work to solve several of these challenges, on Feistel-MiMC, Rescue Prime and Poseidon.

Our results are obtained by writing the challenges as systems of polynomial equations over the large field, and solving them with off-the-shelf tools (`SageMath`, NTL, `Magma`).

Keywords: Arithmetization-oriented hash functions, Poseidon, Feistel-MiMC, Rescue Prime, algebraic cryptanalysis

1 Introduction

On November 1st, challenges for several arithmetization-oriented hash functions over large prime fields were announced on https://www.zkhashbounties.info/, with the goal to solve the CICO problem (Constrained Input - Constrained Output) for the inner permutation.

The challenges, funded by the Ethereum Foundation, targeted reduced functions with the following parameters:

Rescue Prime [SAD20]:

Category	Parameters	Security Level (bits)	Bounty
Easy	N=4, m=3	25	$2,000
Easy	N=6, m=2	25	$4,000
Medium	N=7, m=2	29	$6,000
Hard	N=5, m=3	30	$12,000
Hard	N=8, m=2	33	$26,000

Feistel-MIMC [AGR+16]:

Category	Parameters	Security Level (bits)	Bounty
Easy	r=6	9	$2,000
Easy	r=10	15	$4,000
Medium	r=14	22	$6,000
Hard	r=18	28	$12,000
Hard	r=22	34	$26,000

Poseidon [GKR+21]:

Category	Parameters	Security Level (bits)	Bounty
Easy	RP=3	8	$2,000
Easy	RP=8	16	$4,000
Medium	RP=13	24	$6,000
Hard	RP=19	32	$12,000
Hard	RP=24	40	$26,000

Reinforced Concrete [BGK+21]:

Category	Parameters	Security Level (bits)	Bounty
Easy	p=281474976710597	24	$4,000
Medium	p=72057594037926839	28	$6,000
Hard	p=18446744073709551557	32	$12,000

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:

Category	Parameters	Security Level (bits)	Bounty
~~Easy~~	~~r=22~~	~~18~~	~~$2,000~~
Easy	r=25	20	$4,000
Medium	r=30	24	$6,000
Hard	r=35	28	$12,000
Hard	r=40	32	$26,000

In this work, we present simple algebraic attacks against some of these targets. We model the CICO problem as a system of polynomial equations over the field, and we use off-the-shelf computer algebra tools to solve the system.

2 Solving a System of Polynomial Equations

We first give a quick overview of classical methods to solve a system of polynomial equations over a prime field. We assume that the system has the same number of variables and equations, and that it behaves similarly to a random system, with one solution on average.

2.1 Univariate case

In the univariate case, solving a polynomial system is equivalent to finding the roots of a polynomial $P \in \mathbb{F}_p[X]$ in a finite field of prime characteristic p. For a polynomial P of degree d, finding the roots requires $\mathcal{O}\big(d\log(d)\big(\log(d)+\log(p)\big)\log(\log(d))\big)$ field operations, using the following method. We assume that we can multiply two polynomials of degree d with $\mathcal{O}\big(d\log(d)\log(\log(d))\big)$ field operations using an FFT algorithm.

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

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

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

In particular, finding the roots in the prime field is significantly easier than factoring the polynomial (it is quasi-linear in the degree). In practice, we use the NTL library, and the computation is feasible up to degree roughly $3^{20} \approx 2^{31.7}$ with $p \approx 2^{64}$; we show some performance results in Table 1.

Table 1: Benchmarks of univariate root finding with NTL, using 1 CPU core of Intel Xeon E7-4860.

Degree	3^{11}	3^{12}	3^{13}	3^{14}	3^{15}	3^{16}	3^{17}	3^{18}
Time	20s	75s	3m40s	13m	39m	2h8m	6h15m	22h36m
Memory	110MB	325MB	835MB	2,7GB	7,5GB	22,8GB	64GB	223GB

2.2 Multivariate case

In the multivariate case, finding solutions of a polynomial system can be done by computing a Gröbner basis and converting it to an elimination order. The complexity depends on the number of solutions D in the algebraic closure of $\mathbb{F}_p$; in the generic case D is the product of the degrees of the polynomials in the system (the Bézout bound). The complexity of solving the system is essentially $\mathcal{O}(D^3)$, but it can be reduced to $\mathcal{O}(D^\omega)$ asymptotically using fast linear algebra [FGHR14] (with $2 \leq \omega < 2.3727$ the exponent of linear algebra).

In practice, we use the `Magma` system, and the computation is feasible up to roughly $D = 3^9$ (our experimental results are summarized in Table 2).

Comparing Table 1 with Table 2, we can see that solving a multivariate system is significantly harder than solving a univariate one with the same number of solutions in the algebraic closure. Therefore, we will try to build univariate systems when possible.

Table 2: Benchmarks of multivariate system solving with `Magma`, using 1 CPU core of an Intel Xeon Gold 5218.

D	3^6	3^9
Time	9 s	4 days
Memory	100MB	58GB

3 Algebraic modelization of the challenges

Let p be a prime number, and $t \geq 2$ be some integer. We denote $\{e_i\}_{i<t}$ the canonical basis of $\mathbb{F}_p^t$, so that $e_0 = (1, 0, ..., 0)$, etc. We also denote $\rho_i : \mathbb{F}_p^t \to \mathbb{F}_p$ the function mapping $x = (x_0, ..., x_{t-1})$ to x_i.

Let $u < t$ be an integer, and let $\mathcal{Z}_u$ be the vector space spanned by $\{e_0, ..., e_{t-u-1}\}$. In other words, $\mathcal{Z}_u$ is the set of all the elements of $\mathbb{F}_p^t$ 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 $F : \mathbb{F}_p^t \to \mathbb{F}_p^t$ be a function, and let $u < t$ be an integer. The *CICO problem* consists in finding $x \in \mathbb{F}_p^t$ such that

$$x \in \mathcal{Z}_u \quad \text{and} \quad F(x) \in \mathcal{Z}_u \ .$$

In what follows, we consider the case where $u = 1$. In this case, we use the simpler notation $\mathcal{Z} = \mathcal{Z}_1$.

3.1 Basic Approach

All the challenges proposed have $u = 1$. In this case, the CICO problem for a function $F : \mathbb{F}_p^t \to \mathbb{F}_p^t$ consists in finding $x \in \mathbb{F}_p^t$ such that

$$\rho_{t-1}(x) = \rho_{t-1}\left(F(x)\right) = 0 \ .$$

Let V be a vector of $\mathbb{F}_p^t$. If $V \in \mathcal{Z}_u$, then finding an $\mathbf{X} \in \mathbb{F}_p$ such that $\rho_{t-1} \circ F(\mathbf{X}V) = 0$ is sufficient to solve the CICO problem, and corresponds to finding roots of a univariate polynomial. This simple attack works against some round-reduced variants of Feistel-MiMC.

3.2 A More Advanced Trick

A more sophisticated approach uses two steps.

Let $P = P_0 \circ P_1$ be a permutation of $\mathbb{F}_p^t$. Suppose that there exists two vectors V and G in $\mathbb{F}_p^t$ such that

$$P_0^{-1}(\mathbf{X}V + G) \in \mathcal{Z}$$

for all $\mathbf{X} \in \mathbb{F}_p$. In this case, we can first find $\mathbf{X}$ such that $P_1(\mathbf{X}V + G) \in \mathcal{Z}$. Then, setting $x = P_0^{-1}(\mathbf{X}V + G)$ will yield a solution to the CICO problem, while the solver has to handle a polynomial based on P_1 rather than the full P. This approach is summarized in Figure 1, and we used it against both Poseidon (see Section 5) and Rescue Prime (see Section 6).

4 Attacks Against Round-Reduced Feistel-MiMC

Feistel-MiMC operates on $\mathbb{F}_p^2$ ($t = 2$) using a basic r-round Feistel structure with the i-th round function being $x \mapsto (x + c_i)^3$. In order to build a polynomial system representing the CICO problem, we consider an input state $(P_0, Q_0) = (\mathbf{X}, 0)$ (*i.e.* we use the basic approach of Section 3.1 with $V = (1, 0) \in \mathcal{Z}_1$). Then we evaluate the round function iteratively, as polynomials in $\mathbb{F}_p[\mathbf{X}]$:

$$\begin{aligned} P_0 &= \mathbf{X} & Q_0 &= 0 \\ P_i &= Q_{i-1} + (P_{i-1} + c_i)^3 & Q_i &= P_{i-1} \ . \end{aligned}$$

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.

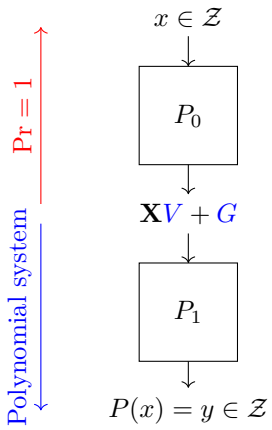

Figure 1: A 2-staged trick.

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

$$d\log(d)\big(\log(d)+\log(p)\big)\log(\log(d)) \;\approx\; 3^{r-1}\times(r-1)\times 1.58\times 64\times\log_2(r-1).$$

We give explicit values for the proposed challenges in Table 3. Detailled time and memory complexity for small instances can be found in Table 1.

Table 3: Complexity of our attack against Feistel-MiMC, compared with the security claims given with the challenges. Time is given for attacks that we have implemented in practice.

Original parameters					**New parameters**			
r	claim	d	complexity	time	r	claim	d	complexity
6	2^{18}	3^5	2^{19}	$<$ 1s	22	2^{36}	3^{21}	2^{47}
10	2^{30}	3^9	2^{26}	1s	25	2^{40}	3^{24}	2^{52}
14	2^{44}	3^{13}	2^{33}	3min40s	30	2^{48}	3^{29}	2^{60}
18	2^{56}	3^{17}	2^{40}	6h15min	35	2^{56}	3^{34}	2^{69}
22	2^{68}	3^{21}	2^{47}		40	2^{64}	3^{39}	2^{77}

5 Attacks Against Round-Reduced Poseidon

The basic approach we described in Section 3.1 works for the easiest instance of Poseidon. For the next one, we need to be a bit more clever, and thus use the technique from Section 3.2. The idea is to decrease the degree and the complexity of the polynomial system by more carefully choosing its variable.

We consider an input state after the S-box layer of the second round of the form $(A^3\mathbf{X}, B^3\mathbf{X}, g)$ and we study the first rounds as shown in Figure 2 (*i.e.* we use $V = (A^3, B^3, 0)$ and $G = (0, 0, g)$).

As in the specification, we use c_i^r to denote the i-th round constant used in round r. We let the linear layer M be such that

$$M^{-1} = \begin{bmatrix} \alpha_0 & \beta_0 & \gamma_0 \\ \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \end{bmatrix}.$$

As a consequence, in Figure 2, the value $(c_2^0)^3$ must satisfy

$$\begin{aligned}(c_2^0)^3 &= \alpha_2(A\mathbf{X}^{1/3} - c_0^1) + \beta_2(B\mathbf{X}^{1/3} - c_1^1) + \gamma_2(g^{1/3} - c_2^1) \\ &= \mathbf{X}^{1/3}\left(\alpha_2 A + \beta_2 B\right) + \gamma_2 g^{1/3} - \gamma_2 c_2^1 - \alpha_2 c_0^1 - \beta_2 c_1^1 \ .\end{aligned}$$

It is the case provided for instance that:

$$\begin{cases} B &= -\frac{\alpha_2 A}{\beta_2} \\ g &= \left(\frac{\gamma_2 c_2^1 + \alpha_2 c_0^1 + \beta_2 c_1^1}{\gamma_2} + (c_2^0)^3\right)^3 \ . \end{cases} \tag{1}$$

As a consequence, if we find a value $\mathbf{X}$ such that the image of $(A^3\mathbf{X}, B^3\mathbf{X}, g)$ 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 Poseidon is mapped to $\mathcal{Z}$.

Therefore, we evaluate the permutation as polynomials in $\mathbb{F}_p[\mathbf{X}]$ starting from the state $(A^3\mathbf{X}, B^3\mathbf{X}, g)$ with A, B, g satisfying System (1), and the CICO problem is equivalent to finding the root of the polynomial corresponding 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.

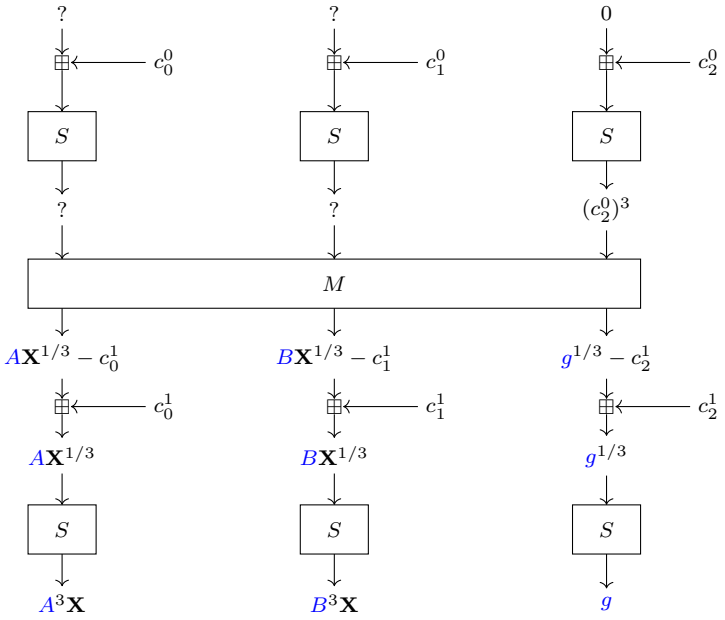

Figure 2: How to bypass 2 non-linear layers of Poseidon.

Complexity Analysis. Poseidon has $r = \mathsf{RF} + \mathsf{RP}$ 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:

$$d\log(d)\big(\log(d) + \log(p)\big)\log(\log(d))\big) \approx 3^{r-2} \times (r-2) \times 1.58 \times 64 \times \log_2(r-2).$$

We give explicit values for the proposed challenges in Table 4. Detailled time and memory complexity for small instances can be found in Table 1. For the instance with $\mathsf{RP} = 13$, the computation took 36 hours on a Xeon E7-4860 v2 using less than a terabyte of RAM, using parallelization available in NTL with 32 cores available. However, most of the time was spend with only one or three cores active; we did not investigate further how NTL uses multiple cores.

Table 4: Complexity of our attack against Poseidon, compared with the security claims given with the challenges. Time is given for attacks that we have implemented in practice.

RP	claim	d	complexity	time
3	2^{45}	3^9	2^{26}	1s
8	2^{53}	3^{14}	2^{35}	13min
13	2^{61}	3^{19}	2^{44}	36h (multiple cores)
19	2^{69}	3^{25}	2^{54}	
24	2^{77}	3^{30}	2^{62}	

6 Attacks Against Round-Reduced Rescue Prime

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 an m-element state ($m = 2$ or $m = 3$) 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 ($m(N+1)$ variables in total). As shown in Figure 3, we can write m equations linking the m 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:

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

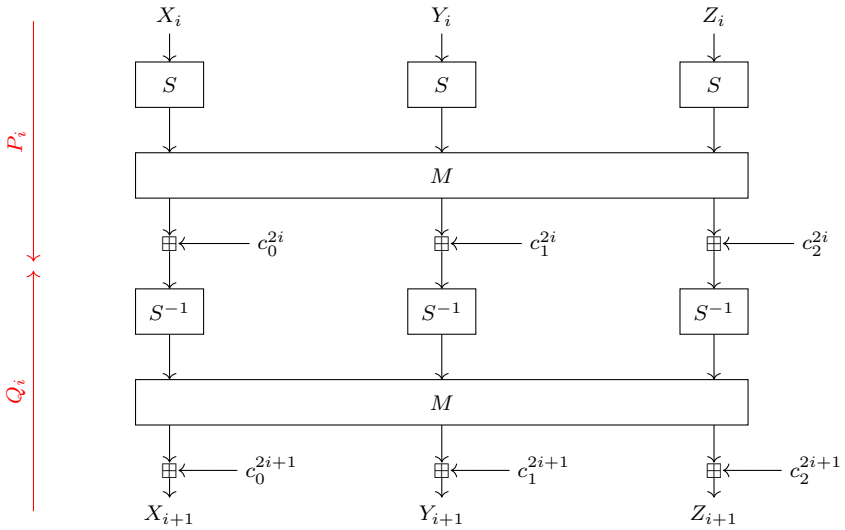

Figure 3: Polynomial equations for one round of Rescue Prime.

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, because we can directly write a degree-3 polynomial of $X_1, Y_1, \ldots$ that must be equal to $S(X_0) = 0$. We can also remove X_N because it is fixed to zero, and we obtain a system of $m(N-1)+1$ equations and $mN-1$ variables.

With $m = 2$, we have the same number of equations and variables. However, with $m \geq 3$ we have more variables than equations, and we can use the trick of Section 3.2 to

obtain a smaller system corresponding to a subset of the solutions with one solution on average.

Bypassing the First Round. Let us repeat the idea described in Section 5 and apply it for Rescue Prime. We consider an input state after the S-box layer of the second round of the form $(A^3\mathbf{X}, B^3\mathbf{X}, g)$ and we study the first rounds as shown in Figure 4 (*i.e.* we use $V = (A^3, B^3, 0)$ 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:

$$C_2^0 = \alpha_2 c_0^0 + \beta_2 c_1^0 + \gamma_2 c_2^0 .$$

As a consequence, using the same notations as above, the value C_2^0, in Figure 4, must satisfy

$$\begin{aligned} C_2^0 &= \alpha_2 A\mathbf{Y}^3 + \beta_2 B\mathbf{Y}^3 + \gamma_2 g^3 \\ &= \mathbf{Y}^3 (\alpha_2 A + \beta_2 B) + \gamma_2 g^3 . \end{aligned}$$

It is the case provided for instance when:

$$\begin{cases} B &= -\frac{\alpha_2 A}{\beta_2} \\ g &= \left(\frac{\alpha_2 c_0^0 + \beta_2 c_1^0 + \gamma_2 c_2^0}{\gamma_2}\right)^{1/3} \end{cases} . \tag{2}$$

It follows that, if we find a value $\mathbf{Y}$ such that the image of $(A^3\mathbf{Y}, B^3\mathbf{Y}, g)$ 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 $\mathcal{Z}$.

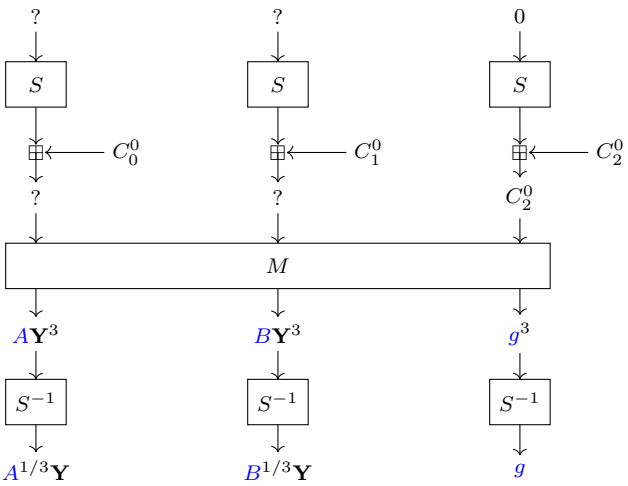

Figure 4: How to bypass the first round of Rescue Prime.

Then, for the remaining $R-1$ rounds, Figure 3 shows how we generate the following polynomial equations to avoid the inverse S-box.

$$\forall j = \{0, 1, 2\},\ P_{i,j}(X_i, Y_i, Z_i) - Q_{i,j}(X_{i+1}, Y_{i+1}, Z_{i+1}) = 0 .$$

Finally, this results in the following system of polynomial equations:

$$\begin{cases} \forall\, 1 \leq i \leq N-1, \forall\, j = \{0,1,2\}, \\ P_{i,j}(X_i, Y_i, Z_i) - Q_{i,j}(X_{i+1}, Y_{i+1}, Z_{i+1}) = 0\ , \end{cases} \tag{3}$$

where $Z_N = 0$ and

$$\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = M \begin{pmatrix} A^{1/3}\mathbf{Y} \\ B^{1/3}\mathbf{Y} \\ g \end{pmatrix} + \begin{pmatrix} c_0^1 \\ c_1^1 \\ c_2^1 \end{pmatrix}\ .$$

This system has $m(N-1)$ variables and $m(N-1)$ equations.

Complexity Analysis. With $m = 3$ 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:

$$D^3 = 3^{9(N-1)}.$$

With $m = 2$ branches and N rounds, we obtain a system of $2N-1$ degree-3 equations with the same number of variables. Therefore, the complexity of solving the system is approximately:

$$D^3 = 3^{6N-3}.$$

We give explicit values for the proposed challenges in Table 5. Detailled time and memory complexity for small instances can be found in Table 2.

Table 5: Complexity of our attack against Rescue Prime, compared with the security claims given with the challenges. Time is given for the attack that we have implemented in practice.

N	m	claim	D	complexity	time
4	3	$2^{37.5}$	3^9	2^{43}	4 days
6	2	$2^{37.5}$	3^{11}	2^{53}	
7	2	$2^{43.5}$	3^{13}	2^{62}	
5	3	2^{45}	3^{12}	2^{57}	
8	2	$2^{49.5}$	3^{15}	2^{72}	

Acknowledgements

We thank Jules Baudrin and Clara Pernot for proof reading a first draft of this manuscript, and Magali Bardet for helpful discussions about solving multivariate systems. This work was partially financially supported by the french ministry of defence – Agence de l'Innovation de Défense (AID).

References

[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 Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

[BGK+21] Mario Barbara, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofneggerk, and Roman Walch. Reinforced concrete: Fast hash function for zero knowledge proofs and verifiable computation. Cryptology ePrint Archive, Report 2021/1038, 2021. https://eprint.iacr.org/2021/1038.

[FGHR14] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. Sub-cubic change of ordering for gröbner basis: a probabilistic approach. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 170–177, 2014.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. POSEIDON: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2020*, pages 1–17. USENIX Association, August 2021.

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Report 2020/1143, 2020. https://eprint.iacr.org/2020/1143.

A Implementation

By default NTL is limited to polynomials of degree at most 2^{24}. In order to generate the largest systems, NTL must be recompiled with the following patch:

```
--- ntl-11.5.1/include/NTL/FFT.h
+++ ntl-11.5.1.patched/include/NTL/FFT.h
@@ -24,15 +24,15 @@
 // Absolute maximum root bound for FFT primes.
 // Don't change this!

-#if (25 <= NTL_FFTMaxRootBnd)
-#define NTL_FFTMaxRoot (25)
+#if (32 <= NTL_FFTMaxRootBnd)
+#define NTL_FFTMaxRoot (32)
 #else
 #define NTL_FFTMaxRoot  NTL_FFTMaxRootBnd
 #endif
 // Root bound for FFT primes.  Held to a maximum
-// of 25 to avoid large tables and excess precomputation,
+// of 32 to avoid large tables and excess precomputation,
 // and to keep the number of FFT primes needed small.
-// This means we can multiply polynomials of degree less than 2^24.
+// This means we can multiply polynomials of degree less than 2^31.
 // This can be increased, with a slight performance penalty.
```

In order to generate the polynomial systems, `SageMath` must also be tweaked to use this version of NTL (for instance by using `LD_PRELOAD`).