Algebraic Attacks on RAIN and AIM Using Equivalent Representations

Fukang Liu¹, Mohammad Mahzoun², Morten Øygarden³, Willi Meier⁴

¹ Tokyo Institute of Technology, Tokyo, Japan
liufukangs@gmail.com
² Eindhoven University of Technology, Netherlands
mail@mahzoun.me
³ Simula UiB, Bergen, Norway
morten.oygarden@simula.no
⁴ FHNW, Windisch, Switzerland
willimeier48@gmail.com

Abstract. Designing novel symmetric-key primitives for advanced protocols like secure multiparty computation (MPC), fully homomorphic encryption (FHE) and zero-knowledge proof systems (ZK), has been an important research topic in recent years. Many such existing primitives adopt quite different design strategies from conventional block ciphers. Notable features include that many of these ciphers are defined over a large finite field, and that a power map is commonly used to construct the nonlinear component due to its efficiency in these applications as well as its strong resistance against the differential and linear cryptanalysis. In this paper, we target the MPC-friendly ciphers AIM and RAIN used for the post-quantum signature schemes Aimer (CCS 2023 and NIST PQC Round 1 Additional Signatures) and Rainier (CCS 2022), respectively. Specifically, we can find equivalent representations of 2-round RAIN and full-round AIM, respectively, which make them vulnerable to either the polynomial method, or the crossbred algorithm, or the fast exhaustive search attack. Consequently, we can break 2-round RAIN with the 128/192/256-bit key in only $2^{111}/2^{170}/2^{225}$ bit operations. For full-round AIM with the 128/192/256-bit key, we could break them in $2^{136}/2^{200}/2^{265}$ bit operations, which are equivalent to about $2^{115}/2^{178}/2^{241}$ calls of the underlying primitives. In particular, our analysis indicates that AIM does not reach the required security levels by the NIST competition.

Keywords: RAIN · AIM · equivalent representation · overdefined system · algebraic attack

1 Introduction

Developing novel symmetric-key primitives with applications to MPC, FHE and ZK has become an important research topic for their real-world impacts. Since the block cipher LowMC [ARS+15] published at EUROCRYPT 2015, there haven been a number of such novel primitives [AGR+16, CHK+21, GLR+20, GKL+22, DKR+22, BBC+23, DEG+18, GAH+23, GLR+20, MJS+16, DGK+21, GØSW+23, HKL+22, CCF+16, DGH+21, AAB+20, AMT22, SLST23, GKR+21]. Especially, some ZK-friendly hash functions like Poseidon [GKR+21] and Rescue [AAB+20] were immediately adopted in real-world blockchains for their high efficiency in this application, and some MPC-friendly primitives like LowMC [ARS+15], Rain [DKR+22] and AIM [KHS+22] have been used to build post-quantum signature schemes with the MPC-in-the-head technique [IKOS07].

While traditional symmetric-key primitives typically work over $\mathbb{F}_2$, many of the new primitives are designed over large finite fields, though some are still defined over $\mathbb{F}_2$, but
with new structures. This poses new challenges in terms of developing generic or dedicated cryptanalysis techniques to understand their security. Since this has been a largely unexplored area, many of these ciphers eventually turned out to be vulnerable against new attacks against their less-understood components, like the guess-and-determine attack on FLIP [DLR16], Gröbner basis attack on Jarvis and Friday [ACG+19], various algebraic attacks on LowMC [LIM21, LSW+22, Din21a, LMSI22, BBDV20, BBVY21], linearization attack on Rasta [LSMI21], higher-order differential attacks on MiMC [EGL+20, BCD+20, BCP23] and Chaghri [LAW+23, LGB+23] over large finite fields, and algebraic attacks on Rubato over rings [GAH+23], just to name a few. After years of effort to understand the security of these novel primitives, there have been some useful cryptanalytic tools developed for designers. However, this does not imply that the designers could exclude all potential attacks for their new proposals, especially when the ciphers are adopting new and less-understood structures.

In this paper, we mainly focus on the latest MPC-friendly ciphers RAIN (CCS 2022) [DKR+22] and AIM (CCS 2023) [KHS+22]. The two ciphers are tailored to specific post-quantum signature schemes Rainier and AIMer built with the MPC-in-the-head technique, respectively. Especially, AIMer is one of NIST PQC Round 1 Additional Signatures\(^1\) that were released in July, 2023. In the signature schemes Rainier and AIMer, the public key is a single plaintext-ciphertext pair, and the secret key is the key for this plaintext-ciphertext pair. Consequently, the security of these two signature schemes relies on the difficulty to recover the secret key of RAIN and AIM with only a single known plaintext-ciphertext pair, respectively. While there are some successful attacks [Din21a, LMSI22, BBDV20, BBVY21] on LowMC in this setting by exploiting the low-degree properties of its 3-bit S-box, RAIN and AIM are quite different from LowMC as their nonlinear components are over large finite fields and their algebraic degree is high or maximal. Therefore, the designers made very aggressive choices for the secure number of rounds for the two ciphers in order to improve the efficiency of the signature schemes. Specifically, it is claimed that 3 rounds are secure for RAIN and the designers also proposed to use 4 rounds to further increase the security margin. Moreover, by the designers’ analysis, even 2 rounds of RAIN cannot be broken. For AIM, although it is different from common block ciphers, we can view it as a 2-round primitive.

Our Contributions. In spite of the high algebraic degree of the nonlinear components of RAIN and AIM, we could identify nontrivial low-degree equivalent representations for 2-round RAIN and full-round AIM, which make them vulnerable to either the polynomial method [Din21a], crossbred algorithm [JV17] or the fast exhaustive search attack [BCC+10]. The main idea is to set up a system of equations in a variable related to the secret key, i.e. they are related by a high-degree equation. After solving the system of equations to recover this variable, the secret key can be then efficiently computed, and the key-recovery attack is achieved. Specifically, we could

- break 2-round RAIN using the polynomial method or the crossbred algorithm with time complexity significantly smaller than \(2^n\) bit operations, as shown in Table 1;
- significantly improve the naive brute force attack on AIM with the fast exhaustive search method\(^2\) to solve a system of low-degree equations, as shown in Table 3.

Organization. In Sect. 2, we briefly introduce the relation between polynomials over \(\mathbb{F}_{2^n}\) and \(\mathbb{F}_2\), and revisit some known techniques to solve the system of multivariate Boolean equations. Then, we show in Sect. 3 how to derive an overdefined system of quadratic Boolean equations from a special form of polynomial equations over \(\mathbb{F}_{2^n}\). In Sect. 4, we

\(^{1}\)https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures

\(^{2}\)In private communication, the AIM team has acknowledged our attack.
will present the algebraic attacks on 2 rounds of RAIN. The fast exhaustive search attack on full-round AIM is given in Sect. 5. Finally, the paper is concluded in Sect. 6.

2 Solving Multivariate Nonlinear Boolean Equations

It is well-known that solving multivariate nonlinear equations is NP-hard, even if the equations are over \( \mathbb{F}_2 \), i.e. Boolean equations. In algebraic attacks on cryptographic primitives, the attackers first model the to-be-solved problem (e.g., key recovery or finding a preimage/collision) with a system of nonlinear equations, and then solve the equation system with generic techniques like Gröbner basis method \([\text{Buc65, Fau99, Fau02}]\), XL algorithm \([\text{CKPS00}]\), fast exhaustive search (FES) \([\text{BCC10}]\), crossbred algorithm \([\text{JV17}]\) or polynomial method \([\text{LPT17, BKW19, Din21b, Din21a}]\). Developing generic equation-solving techniques is always challenging. In most cases, how to model the problem with better algebraic equations is crucial to improve the algebraic attack because attackers can exploit some algebraic structures of the primitives to set up equations that can be efficiently solved.

Since our target primitives RAIN \([\text{DKR+22}]\) and AIM \([\text{KHS+22}]\) are both defined over the finite field \( \mathbb{F}_{2^n} \), and there exists an isomorphism between \( \mathbb{F}_{2^n} \) and \( \mathbb{F}_2^n \), e.g., see Chapter 6 of \([\text{RHPScKK07}]\), we only focus on Boolean equations in this paper. Specifically, each polynomial \( p(y_1, \ldots, y_t) \) in the polynomial ring \( \mathbb{F}_{2^n}[y_1, \ldots, y_t] \) can be written as

\[
p(y_1, \ldots, y_t) = \sum_{i_1=0}^{2^n-1} \cdots \sum_{i_t=0}^{2^n-1} u_{i_1, \ldots, i_t} y_{i_1}^1 \cdots y_{i_t}^t,
\]

where \( u_{i_1, \ldots, i_t} \in \mathbb{F}_{2^n} \) is the coefficient. Given any irreducible polynomial for \( \mathbb{F}_{2^n} \), the polynomial \( p \) can be equivalently described by \( n \) vectorial Boolean polynomials \( p_1, \ldots, p_n \) in \( nt \) Boolean variables, and the algebraic degree of each Boolean polynomial \( p_i \) is

\[
\text{Deg}(p) = \max \left\{ \sum_{i=1}^t \text{Hw}(i_j) : u_{i_1, \ldots, i_t} \neq 0 \right\},
\]

where \( \text{Hw}(i) \) denotes the Hamming weight of the integer \( i \), i.e. the number of 1 in the binary representation of \( i \). Indeed, we can have the following formal definition of the algebraic degree of a polynomial in \( \mathbb{F}_{2^n}[y_1, \ldots, y_t] \):

**Definition 1.** Let

\[
p(y_1, \ldots, y_t) = \sum_{i_1=0}^{2^n-1} \cdots \sum_{i_t=0}^{2^n-1} u_{i_1, \ldots, i_t} y_{i_1}^1 \cdots y_{i_t}^t,
\]

be a polynomial in \( \mathbb{F}_{2^n}[y_1, \ldots, y_t] \) where \( u_{i_1, \ldots, i_t} \in \mathbb{F}_{2^n} \) is the coefficient, the algebraic degree of \( p \) denoted by \( \text{Deg}(p) \) is defined as

\[
\text{Deg}(p) = \max \left\{ \sum_{i=1}^t \text{Hw}(i_j) : u_{i_1, \ldots, i_t} \neq 0 \right\}.
\]

In other words, for a system of equations defined over \( \mathbb{F}_{2^n} \), we can first convert them into a system of Boolean equations with known algebraic degree, and then solve them with generic equation-solving techniques developed for Boolean equations. Let us consider \( m \) Boolean polynomials \( f_1, \ldots, f_m \) in \( n (n \leq m) \) variables \( x_1, \ldots, x_n \), and their algebraic degree is upper bounded by \( d \). For the following system of \( m \) equations:

\[
 f_1(x_1, \ldots, x_n) = 0, \quad \ldots, \quad f_m(x_1, \ldots, x_m) = 0,
\]

there are some known techniques to solve them and we introduce them one by one.
2.1 Fast Exhaustive Search

The fast exhaustive search method is basically based on an efficient way to evaluate a Boolean polynomial over \( \{0, 1\}^n \), i.e. how to efficiently compute \( f_i(x_1, \ldots, x_n) \) for all \( (x_1, \ldots, x_n) \in \{0, 1\}^n \). Since the algebraic degree of each \( f_i \) is upper bounded by \( d \), there are \( \binom{n}{\leq d} \) possible terms in \( f_i(x_1, \ldots, x_n) \) where \( \binom{n}{\leq d} = \sum_{i=0}^{d} \binom{n}{i} \). Therefore, the naive polynomial evaluation would require about \( 2^n \cdot \binom{n}{\leq d} \) bit operations for each \( f_i \).

In [BCC⁺10], the authors found that evaluating \( f_i \) over \( \{0, 1\}^n \) only requires \( d \cdot 2^n \) bit operations after an initialization phase of complexity \( O(n^{2d}) \) (see Theorem 1 in [BCC⁺10]). Indeed, according to the explanation in Section 4 of [BCC⁺10], the term \( n^{2d} \) comes from \( n^d \) times of evaluations of the polynomial \( f_i \), which is itself upper bounded by \( n^d \). This suggests the complexity of such an initialization phase is indeed about \( n^d \cdot \binom{n}{\leq d} \) bit operations.

With such an efficient way to evaluate \( f_i \), given \( n \) polynomials of algebraic degree \( d \), the attackers can first evaluate \( n_1 \) polynomials \( f_1, \ldots, f_{n_1} \) in parallel. At this phase, it is expected to generate \( 2^{n-n_1} \) candidate solutions. For each such candidate solution, it can be further checked against the remaining polynomials with the naive polynomial evaluation aided with an early-aborting strategy. In this way, the time complexity to find the solution is estimated as about \( 4d \cdot \log_2 n \cdot 2^n \) bit operations (see Theorem 2 and its generalization at Section 5 of [BCC⁺10]). The memory complexity is estimated as \( n \cdot \binom{n}{\leq d} \) bits, which is used to store the polynomials. Note that the time complexity of the initialization phase is upper bounded by \( n_1 \cdot n^d \cdot \binom{n}{\leq d} \leq n \cdot n^d \cdot \binom{n}{\leq d} \) bit operations according to the above careful analysis, and this is not taken into account in \( 4d \cdot \log_2 n \cdot 2^n \).

At EUROCRYPT 2021, Dinur proposed a memory-efficient Möbius transform [Din21a] to evaluate a Boolean polynomial \( f_i(x_1, \ldots, x_n) \) of algebraic degree \( d \) over \( \{0, 1\}^n \). This method does not require the costly initialization phase of time complexity \( O(n^{2d}) \). Moreover, the time complexity and memory complexity of this method are \( d \cdot 2^n \) bit operations and \( n \cdot \binom{n}{\leq d} \) bits, respectively. Hence, it can be trivially applied to the fast exhaustive search and the whole time complexity is kept the same, i.e. it is still \( 4d \cdot \log_2 n \cdot 2^n \) bit operations. The memory complexity is upper bounded by \( n \cdot n \cdot \binom{n}{\leq d} \) bits.

2.2 Polynomial Method

The polynomial method to solve \( n \) multivariate Boolean equations of algebraic degree \( d \) in \( n \) variables was first proposed at SODA 2017 [LPT⁺17]. There have been 3 improved variants [BKWi19,Din21b,Din21a] and we focus on the latest variant [Din21a] proposed by Dinur at EUROCRYPT 2021 since it gives the accurate time complexity in bit operations. The general idea is to first find the solutions to a smaller number of polynomial equations from these \( n \) polynomial equations, and then check the solutions against the remaining equations. The technique mainly exists in how to efficiently enumerate the the solutions of that small system of equations by the polynomial interpolation and evaluation. We refer the interested readers to [Din21a,LMSI22] for the detailed explanation. For a system of \( n \) Boolean equations of algebraic degree \( d \) (\( d \geq 2 \)) in \( n \) variables, the time and memory complexity to solve it are estimated as \( n^2 \cdot 2^{(1−1/2 \cdot 7d)n} \) bit operations and \( n^2 \cdot 2^{(1−1/35d)n} \) bits, respectively. In the case \( d = 2 \), the time and memory complexity are \( n^2 \cdot 2^{81.5n} \) bit operations and \( n^2 \cdot 2^{0.63n} \) bits, respectively.

Due to the relatively low time complexity of the polynomial method, it has become an important tool to evaluate the resistance against algebraic attacks, especially when it is possible to construct low-degree equations. For the downsides of this method, apart from its requirement of huge memory, it also cannot take advantage of the feature of the overdefined system of equations. Specifically, even if we have \( m > n \) equations in \( n \) variables, the polynomial method will give the same time complexity as in the case of \( n \) equations.
2.3 Crossbred Algorithm

The idea behind Joux and Vitse’s Crossbred algorithm [JV17] for finding solutions to a polynomial system $F$ of $m$ quadratic equations $f_1 = 0, \ldots, f_m = 0$ in $n$ variables, is to derive a set of polynomials in the ideal of $F$ that is easy to solve once certain variables have been fixed. The algorithm considers three integer inputs $(D, n_D, d_1)$. In a preprocessing step reminiscent of the XL algorithm [CKPS00], the original polynomials of $F$ are extended to the degree $\leq D$ Macaulay matrix $M_{\leq D}(F)$. This is the matrix whose rows are given by $m \cdot f$ where $f \in F$ and $m$ is a monomial of degree $\leq D - 2$, and the columns represent all monomials of degree $\leq D$. This matrix is then reduced in a way such that a subsystem of polynomials whose degrees are at most $d_1$ in the first $n_D$ variables. In the linearization step, one iterates over the possible values of the last $n - n_D$ variables, and solves the corresponding degree-$d_1$ subsystem in the first $n_D$ variables. In particular, when $d_1 = 1$, this means to solve a linear system in $n_D$ variables once the last $n - n_D$ variables have been assigned.

Choosing parameters. In general, it is not easy to choose the optimal values of parameters $(D, n_D, d_1)$. In practice, $d_1 = 1$ is typically chosen, and we will focus on this choice as well. To determine $(D, n_D)$, it is often assumed that the underlying polynomial system behaves as a generic system, in that the cancellations between polynomials are minimal. This is also related to the notion of semi-regularity from [BFSY]. Under this assumption, one is able to accurately predict the rank of $M_{\leq D}(F)$ denoted by $\text{Rank}(M_{\leq D}(F))$. Thus, for any given $D$, one can choose the largest corresponding $n_D$ by counting monomials such that there exists a linear subsystem of equations in the first $n_D$ variables. More precisely, let $\text{Mon}_{n,D}(i)$, denote the number of degree $\leq D$-monomials in $n$ variables that have degree $\geq 2$ in the first $i < n$ variables. We then expect to find $M_{\leq D}(F) - \text{Mon}_{n,D}(i)$ equations that are linear in the first $i$ variables. Valid choices of $n_D$ are then any number $i$ that yields $\geq i$ such equations. The pair $(D, n_D)$ can then be chosen so that the running time of the preprocessing and linearization steps are minimized. Indeed, such a way to choose $(D, n_D)$ has, for instance, been adopted in Section 3.4 of [BMSV22].

Unfortunately, the polynomial systems we will work with in Section 4.3 will contain more cancellations than generic systems. Hence the usual estimates outlined above cannot be applied directly. Instead, for small values $D$, we will investigate the rank of $M_D(F)$ for these particular systems, and derive optimal choices of $n_D$ thereafter.

3 Overdefined Systems of Boolean Equations

At ASIACRYPT 2002, Courtois et al. revealed that it was possible to construct an overdefined system of Boolean equations for the S-box of AES [CP02]. Although the algebraic attack on AES in this paper was later believed to be too optimistic or even wrong, it provides a new perspective to analyze AES with algebraic techniques. In our algebraic attacks on RAIN, we rely on a quite similar idea to set up overdefined systems of Boolean equations.

Specifically, according to [CP02], for the inverse function $y = x^{-1}$ over $\mathbb{F}_{2^n}$ where $x \neq 0$, we can derive the following 5 equations over $\mathbb{F}_{2^n}$:

$$xy = 1, \quad x^2y = x, \quad xy^2 = y, \quad x^4y = x^3, \quad xy^4 = y^3.$$  

Due to the isomorphism between $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$, given any irreducible polynomial over $\mathbb{F}_{2^n}$, these 5 equations over $\mathbb{F}_{2^n}$ can be equivalently represented as $5n$ Boolean equations of algebraic degree 2, and these $5n$ quadratic Boolean equations are linearly independent [CL04].
For the S-box $y = S(x)$ of AES and RAIN defined in Equation 1:

\[
\begin{cases}
    y = x^{-1}, & \text{for } \forall x \in \mathbb{F}_{2^n}, x \neq 0, \\
    y = 0, & \text{for } x = 0,
\end{cases}
\]

(1)

it is clear that $xy = 1$ does not hold for the input $x = 0$. However, this does not affect the algebraic attack too much when using these $5n$ Boolean equations. Specifically, if $n$ is relatively large, the input $x = 0$ of the S-box occurs with an extremely low probability $2^{-n}$ when $x$ is a uniform random variable. In this case, we can simply repeat the attack several times. Moreover, in some cases, we can efficiently check whether $x = 0$ holds before using these $5n$ Boolean equations. Therefore, we simply say there are $5n$ quadratic equations to equivalently describe the S-box defined in Equation 1.

### 3.1 Generating More Equations

In the above, we mainly discussed how to generate more equations from the equation $xy = 1$ over $\mathbb{F}_{2^n}$. In [CL04], the authors also discussed how to generate more quadratic equations for different power maps over $\mathbb{F}_{2^n}$ with the Gold exponent [Gol68] and Kasami exponent [Kas71], respectively. In this section, we show how to generate more quadratic equations from a special form of equations that appear in our cryptanalysis of RAIN.

Specifically, we consider 3 polynomials $P_1(x), P_2(x), P_3(x) \in \mathbb{F}_{2^n}[x]$ such that

\[
P_1(x)P_2(x) + P_3(x) = 0,
\]

(2)

where $\text{Deg}(P_1) = \text{Deg}(P_2) = \text{Deg}(P_3) = 1$. Then, we can generate the following 3 equations

\[
\begin{align*}
P_1(x)P_2(x) + P_3(x) &= 0, \\
P_1(x)(P_2(x))^2 + P_2(x)P_3(x) &= 0, \\
(P_1(x))^2P_2(x) + P_1(x)P_3(x) &= 0.
\end{align*}
\]

By the definition of the algebraic degree of a polynomial in $\mathbb{F}_{2^n}[x]$ and the property that $(x+y)^2 = x^2 + y^2$ for $\forall x, y \in \mathbb{F}_{2^n}$, we immediately obtain that these 3 equations correspond to $3n$ Boolean equations of algebraic degree upper bounded by 2.

For convenience, these $3n$ Boolean equations are all assumed to be quadratic since for most cases of interest they are indeed quadratic. We will not discuss the sufficient conditions to make these $3n$ quadratic Boolean equations linearly independent. Instead, we only feel interested how to generate more quadratic equations and expect that they are linearly independent. In the actual attacks, we will first test whether these $3n$ quadratic equations are linearly independent by computing the rank of the coefficient matrix formed by these quadratic equations where each quadratic term is viewed as a new independent variable.

### 4 Algebraic Cryptanalysis of RAIN

In this section, we first briefly recall the description of the MPC-friendly cipher RAIN proposed at CCS 2022 [DKR+22]. Then, we present the algebraic cryptanalysis of 2 and 3 rounds of RAIN, respectively, though the 3-round attack eventually failed. Although the successful attack only reaches 2 rounds, the designers choose the total number rounds of RAIN as either 3 or 4. This indicates that there is a small security margin if using 3 rounds of RAIN. It should be emphasized that the designers have performed thorough analysis of RAIN, and that they also performed experiments of the Gröbner basis attack on 2-round RAIN. While they were not able to find a specific attack on 2-round RAIN, they only claim security of RAIN for more than 3 rounds. Similar to [DKR+22], we denote 2 and 3 rounds of RAIN by RAIN$_2$ and RAIN$_3$, respectively.
4.1 Description of RAIN

The $r$-round RAIN is depicted in Figure 1, where $k$ is the secret key and $(c_1, \ldots, c_r)$ are the round constants. The S-box $y = S(x)$ is the inverse function over $\mathbb{F}_{2^n}$, i.e.

$$\begin{cases} y = x^{-1} = \frac{1}{x}, & \text{for } \forall x \in \mathbb{F}_{2^n}, x \neq 0 \\ y = 0, & \text{for } x = 0 \end{cases}$$

Or equivalently, the S-box is defined by the power map:

$$x \mapsto x^{2^{n-2}}, \quad \text{for } \forall x \in \mathbb{F}_{2^n}.$$  

![Figure 1: The $r$-round RAIN](image)

For the linear layers, they are randomly generated and the designers have fixed their choices. Indeed, the binary matrix $M_i$ can also be transformed to an $\mathbb{F}_2$-linearized polynomial by interpolation. Abusing the notation, we can write $M_i(x)$ as

$$M_i(x) = \sum_{j=0}^{n-1} a_{i,j}x^{2^j},$$

where $(a_{i,0}, \ldots, a_{i,n-1}) \in \mathbb{F}_{2^n}^n$ are known constants. In the design, it has been ensured that for each $M_i$, all the corresponding coefficients of the $\mathbb{F}_2$-linearized polynomial are non-zero, i.e. $a_{i,j} \neq 0$ for $j \in [0, n-1]$.

Since the security of RAIN is limited to the case when the attacker can only know one plaintext-ciphertext pair under the same key, the designers choose $r \in \{3, 4\}$. As an attacker, the goal is thus to recover the secret key $k$ from 1 known input-output pair $(s_0, s_r)$. Indeed, in the signature scheme Rainier built on RAIN [DKR+22], $k$ is the secret key while $(s_0, s_r)$ is the public key. Therefore, the above attack is directly related to the security of Rainier.

4.2 Low-degree Representation for RAIN$_2$

The designers of RAIN claim that the polynomial method [Din21a] is infeasible because the maximal algebraic degree is achieved after only one round due to the inverse function. Although the claim for the inverse function is true, we point out that it is feasible to construct a low-degree equation system to equivalently describe RAIN$_2$. Due to this low-degree representation, the polynomial method directly breaks RAIN$_2$ with the 128/192/256-bit key with only about $2^{118}/2^{171}/2^{224}$ bit operations, respectively.

Specifically, as shown in Figure 2, we introduce a variable $v_1 \in \mathbb{F}_{2^n}$ to represent the internal state after the first S-box. Given the known pair $(s_0, s_2)$, we aim to set up a low-degree equation system only in $v_1$. Solving this equation system will allow us to recover $v_1$ and then the secret key $k$ can be trivially recovered via:

$$k = v_1^{-1} + s_0 + c_1.$$

First, we consider whether $v_1 = 0$. In this case, we have $k = s_0 + c_1$ and it can be trivially verified. Similarly, we can trivially verify whether $k + s_2 = 0$. Therefore, in the following, we always assume that

$$k + c_1 + s_0 \neq 0, \quad k + s_2 \neq 0.$$
We observe that it is possible to construct an overdefined system of quadratic Boolean equations in \( n \) variables. Then, according to \( M_1 \) and the last S-box, we have

\[
M_1(v_1) = \frac{1}{v_1} + s_0 + c_1 + c_2 + \frac{1}{v_1} + s_0 + c_1 + s_2.
\]

For convenience, let

\[
t_0 = s_0 + c_1 + c_2, \quad t_1 = s_0 + c_1 + s_2.
\]

Then, \( t_0, t_1 \) are known constants and we have

\[
M_1(v_1) = \frac{1}{v_1} + t_0 + \frac{1}{v_1 + t_1} - \frac{1}{v_1} + t_0 + \frac{v_1}{1 + t_1 v_1}.
\]

Based on this, we further have

\[
v_1(1 + t_1 v_1)M_1(v_1) = 1 + t_1 v_1 + t_0 v_1(1 + t_1 v_1) + v_1^2.
\]

By simplifying the equation, we have

\[
v_1 M_1(v_1) + t_1 v_1^2 M_1(v_1) = 1 + t_1 v_1 + t_0 v_1 + t_0 t_1 v_1^2 + v_1^2.
\]

Therefore, we can obtain \( n \) quadratic Boolean equations in \( v_1 \). Based on Dinur’s algorithm for the polynomial method [Din21a], we can recover \( v_1 \) in about \( n^2 \cdot 2^{0.815n} \) bit operations. Our results are summarized in Table 1.

### 4.3 Analysis of RAIN\(_2\) with Low-memory Complexity

We observe that it is possible to construct an overdefined system of quadratic Boolean equations for RAIN\(_2\), and solve it with the Crossbred algorithm (Section 2.3). We focus on the choices of \( d_1 = 1 \), and \( D = 2, 3, 4 \). Note that the case \( D = 2 \) corresponds to the simplified crossbred algorithm used in [BDT22, WWF+21, LMSI22].

Specifically, we have

\[
F(v_1) = (v_1 + t_1 v_1^2)M_1(v_1) + 1 + t_1 v_1 + t_0 v_1 + t_0 t_1 v_1^2 + v_1^2 = 0. \quad (4)
\]

Note that this is a Type-1 equation, of the same form as Equation (2), hence we can derive

\[
G(v_1) = M_1(v_1)F(v_1)
= (v_1 + t_1 v_1^2)(M_1(v_1))^2 + (1 + t_1 v_1 + t_0 v_1 + t_0 t_1 v_1^2 + v_1^2)M_1(v_1) = 0, \quad (5)
\]

\[
H(v_1) = (v_1 + t_1 v_1^2)F(v_1)
= (v_1 + t_1 v_1^2)^2M_1(v_1) + (1 + t_1 v_1 + t_0 v_1 + t_0 t_1 v_1^2 + v_1^2)(v_1 + t_1 v_1^2) = 0. \quad (6)
\]

which again correspond to \( 2n \) quadratic Boolean equations in \( v_1 \). Thus we have \( n + 2n = 3n \) Boolean equations in \( n \) variables.
Number of linearly independent equations. Recall from Section 2.3 that we are interested in understanding the number of linearly independent Boolean equations of a certain low degree $D$ that we can form from the system $F$ generated by $F$, $G$, and $H$. This is equivalent to estimating the rank of the Macaulay matrix $M_{\leq D}(F)$ for a sufficiently large $n$. We have experimentally verified that the $3n$ quadratic equations forming $F$ are linearly independent, and thus

$$\text{Rank}(M_{\leq 2}(F)) = 3n.$$  

(7)

At degree $D = 3$, we note that there are $3n^2$ possible polynomials of the form $x_i f_j$, where $x_i$ is a Boolean unknown variable and $f_j$ is a polynomial in $F$, but not all of these will be linearly independent. Indeed, similar to the analysis performed in [OFRC20, Section 4], this can at least be partially understood upon examination of the extension field description of Equations (4)–(6). Denote $L = 1 + t_1v_1 + t_0v_1 + t_0t_1v_1^2 + v_1^2$, and let $F, G, H, M_1$ be as above. Then we have the cancellations associated with

$$M_1 F + G = 0, \quad (v_1 + t_1v_1^2) F + H = 0,$$

$$(v_1 + t_1v_1^2) G + F^2 + LF = 0, \quad M_1 H + F^2 + LF = 0,$$

$$(v_1 + t_1v_1^2) H + (v_1 + t_1v_1^2)^2 F = 0, \quad M_1 G + M_1^2 F = 0.$$

Each of these six cancellations will correspond to a sum of the form $\sum x_i f_j$, which symbolically sums to zero, and hence show up as $6n$ linearly dependent rows in $M_{\leq 3}(F)$. In fact, experiments show that $M_{\leq 3}(F)$ has rank $3n(n+1) - 8n$, meaning that there are $2n$ more cancellations that we are unable to explain. Summing up, this means that we have

$$\text{Rank}(M_{\leq 3}(F)) = n(3n - 8) + \text{Rank}(M_{\leq 2}(F)).$$  

(8)

At degree $D = 4$ we will also have to take into account the trivial syzygies in the polynomial system, i.e. cancellations of the form $f_i f_j + f_j f_i = 0$ and $f_i^2 + f_i = 0$. If these were the only cancellations in the system at degree 4 (as is e.g. the case for a semi-regular system), we would have counted the number of linearly independent equations at degree 4 as $3n(\binom{n}{2}) - (\binom{3n+1}{2})$. However, we know from Equation (8) that there is a correction term of $8n$ at degree 3 which will likely turn into $8n^2$ at degree 4. Finally, experiments show that there is a final correction term of $17n$, which we are unable to explain. Summing up, we find the following formula for the rank of the Macaulay matrix at degree 4:

$$\text{Rank}(M_{\leq 4}(F)) = 3n \binom{n}{2} - \binom{3n + 1}{2} - 8n^2 + 17n + \text{Rank}(M_{\leq 3}(F)).$$  

(9)

Complexity evaluation for the Crossbred algorithm with $D = 2, 3, 4$. In order to choose $n_D$ as described in Section 2.3, we first need $\text{Mon}_{n,D}(i)$, i.e. the number of degree $\leq D$-monomials in $n$ variables that have degree $\geq 2$ in the first $i < n$ variables. This number is listed below for $D = 2, 3, 4$:

$$\text{Mon}_{n,D}(i) = \begin{cases} 
\binom{i}{2} & \text{for } D = 2, \\
\binom{i}{2} + (n-i)\binom{i}{2} + \text{Mon}_{n,2}(i) & \text{for } D = 3, \\
\binom{i}{2} + (n-i)\binom{i}{2} + \binom{n}{2} - \binom{i}{2} + \text{Mon}_{n,3}(i) & \text{for } D = 4.
\end{cases}$$  

(10)

At the preprocessing step of the Crossbred algorithm, we need to find a parameter $n_D$ and eliminate $\text{Mon}_{n,D}(n_D)$ monomials with Gaussian elimination. For these relatively small choices of $D$, the time complexity of the preprocessing step is negligible when compared to the latter linearization step. Specifically, the cost of the preprocessing step is equal
to the cost to perform Gaussian elimination on the Macaulay matrix $M \leq D(\mathcal{F})$ of size \( \binom{\leq D - 2}{\leq D} \cdot 3n \times \binom{n}{\leq D} \). Thus it is optimal to choose $n_D$ as large as possible, that is, for $D = 2, 3, 4$, we define the optimal $n_D$ as

$$n_D = n_D(\mathcal{F}) = \max \{ i \in \mathbb{Z}_{>0} \mid \text{Rank}(M_{\leq D}(\mathcal{F})) - \text{Mon}_{n,D}(i) \geq i \}.$$  \hspace{1cm} (11)

This allows us to construct at least $n_D$ equations that are linear in $n_D$ variables after guessing $n - n_D$ variables, while at the same time keeping $n - n_D$ minimal. In this way, the time complexity to solve the target equation system is estimated as $2^{n - n_D} n_D^3$ bit operations, i.e. the time needed to solve a linear system in $n_D$ variables $2^{n - n_D}$ times.

For the memory complexity, we estimate it as the memory to store all the polynomial equations. Specifically, it is estimated as

$$\left( \binom{n}{\leq D - 2} \cdot m \right) \times \binom{n}{\leq D}$$

bits of memory\(^3\), as the corresponding Macaulay matrix $M_{\leq D}(\mathcal{F})$ has $\binom{n}{\leq D - 2} \cdot 3n$ rows and $\binom{n}{\leq D}$ columns. The resulting time and memory complexities for the RAIN\(_2\) parameters are given in Table 1.

### Table 1: Cost analysis of various methods for solving RAIN\(_2\)

<table>
<thead>
<tr>
<th>Method</th>
<th>$n$</th>
<th>$n_D$</th>
<th>Time (bits)</th>
<th>Memory (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial Method</td>
<td>128</td>
<td>-</td>
<td>$2^{118}$</td>
<td>$2^{95}$</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>-</td>
<td>$2^{172}$</td>
<td>$2^{136}$</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>-</td>
<td>$2^{225}$</td>
<td>$2^{177}$</td>
</tr>
<tr>
<td>Crossbred $D = 2$</td>
<td>128</td>
<td>27</td>
<td>$2^{115}$</td>
<td>$2^{22}$</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>33</td>
<td>$2^{174}$</td>
<td>$2^{23}$</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>38</td>
<td>$2^{234}$</td>
<td>$2^{25}$</td>
</tr>
<tr>
<td>Crossbred $D = 3$</td>
<td>128</td>
<td>30</td>
<td>$2^{113}$</td>
<td>$2^{45}$</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>36</td>
<td>$2^{172}$</td>
<td>$2^{17}$</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>41</td>
<td>$2^{331}$</td>
<td>$2^{40}$</td>
</tr>
<tr>
<td>Crossbred $D = 4$</td>
<td>128</td>
<td>32</td>
<td>$2^{111}$</td>
<td>$2^{45}$</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>38</td>
<td>$2^{170}$</td>
<td>$2^{50}$</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>44</td>
<td>$2^{228}$</td>
<td>$2^{52}$</td>
</tr>
</tbody>
</table>

**Experimental verification.** Our formulas for $\text{Rank}(M_{\leq D}(\mathcal{F}))$ at $D \in \{2, 3, 4\}$ specified in Equations (7)–(9) have been experimentally verified using the built-in algorithm in MAGMA \[BCP97\] for computing the Hilbert series of homogenized systems $\mathcal{F}$ for smaller values of $n$. In particular, $D = 2, 3$ have been tested for $n$ in the range of [25, 40], whereas $D = 4$ has been tested for $n \in [29, 40]$ (note that Equation (9) is negative for $n < 29$).

\(^3\)We note that the time and memory complexity of the preprocessing step to handle the Macaulay matrix $M_{\leq D}(\mathcal{F})$ can be further optimized with advanced implementation techniques. Note that eliminating $\text{Mon}_{n,D}(n_D)$ many monomials can be viewed as performing Gaussian elimination on a $\text{Rank}(M_{\leq D}) \times \text{Mon}_{n,D}(n_D)$ submatrix of the Macaulay matrix, and hence we can consider a $\text{Rank}(M_{\leq D}) \times \text{Mon}_{n,D}(n_D)$ submatrix. After this step, we can build each of the $n_D$ linear equations by summing at most $\text{Rank}(M_{\leq D})$ polynomial equations, which means we do not need to perform the Gaussian elimination on the full Macaulay matrix $M_{\leq D}(\mathcal{F})$ in order to get the desired $n_D$ linear equations. Moreover, the memory complexity can further be reduced by using the Wiedemann algorithm for finding the kernel, as described in \[BMSV22, Section 3.4\]. Still, we prefer to present the numbers from the naïve method for simplicity.
4.4 Analysis of RAIN₃ Using the Equivalent Representation

It is natural to ask whether it is possible to extend this attack to 3 or more rounds. However, this seems infeasible. Specifically, if we work in a similar way, we have

\[
(M_1(v_1) + \frac{1}{v_1} + t_0) \left( \frac{1}{v_1} + s_0 + c_1 + c_3 + M_2^{-1}\left( \frac{1}{v_1} + s_0 + c_1 + s_3 \right) \right) = 1
\]

Let

\[
t_2 = s_0 + c_1 + c_3, \quad t_3 = s_0 + c_1 + s_3.
\]

Then, we have

\[
\left( M_1(v_1) + \frac{1}{v_1} + t_0 \right) \left( \frac{1}{v_1} + t_2 + M_2^{-1} \left( \frac{v_1}{1 + t_3v_1} \right) \right) = 1,
\]

\[
\iff \quad \left( v_1 M_1(v_1) + 1 + t_0 v_1 \right) \left( 1 + t_2 v_1 + v_1 M_2^{-1} \left( \frac{v_1}{1 + t_3v_1} \right) \right) = v_1^2.
\]

In this case, we cannot further expand \( M_2^{-1} \left( \frac{v_1}{1 + t_3v_1} \right) \). Otherwise, we could only get an equation system of algebraic degree \( n - 1 \) in terms of \( v_1 \) because we need to multiply the term

\[
\prod_{i=0}^{n-1} (1 + t_3v_1)^{2^i}
\]

in both sides of the equation to clear all denominators.

Another interesting observation based on the above analysis is that if there are only a few monomials in \( M_1(x) \) or \( M_2^{-1}(x) \), we can still construct a low-degree representation for RAIN₃. For example, if there are only \( \ell \) monomials in \( M_2^{-1}(x) \), i.e. there are only \( \ell \) nonzero coefficients in \( (a_{2,0}', \ldots, a_{2,n-1}') \) of the \( \mathbb{F}_2 \)-linearized polynomial

\[
M_2^{-1}(x) = \sum_{i=0}^{n-1} a_{2,i}'x^{2^i},
\]

by clearing all the denominators, we can get \( n \) Boolean equations of algebraic degree upper bounded by \( \ell + 2 \), which is caused by \( v_1 M_1(v_1) \cdot v_1 M_2^{-1} \left( \frac{v_1}{1 + t_3v_1} \right) \).

The reason to consider 2 cases \( M_1(x) \) and \( M_2^{-1}(x) \) is simple. First, the encryption and decryption of RAIN₃ has the same structure due to the inverse function. Second, we can introduce a variable to denote the input of the last S-box rather than the output of the first S-box, and the equivalent representation only in this variable still has the same form. However, the designers have fixed these polynomials and all the coefficients are nonzero, which makes the above attack infeasible.

5 Algebraic Cryptanalysis of AIM

It is found that there also exists a low-degree representation for full-round AIM. Although the designers also reported a similar observation in the Appendix of [KHS⁺22], they only considered the lower bound of the algebraic degree of this equivalent polynomial representation. What is worse, they only treated the polynomial method [Din21a] as the only threat for this equivalent representation, which requires more than \( 2^n \) bits of memory even for this lower bound. In this section, we perform a deep study on this equivalent representation and give its accurate algebraic degree. By using the fast exhaustive search method [BCC⁺10], we significantly reduce the cost to find the secret key of AIM via brute force.
5.1 Description of AIM

The general construction of AIM can be described as follows:

\[ z_i = k^{2^{e_i} - 1} \text{ for } i \in [1, m - 1], \]
\[ w = \sum_{i=1}^{m-1} B_i(z_i), \]
\[ y = w^{2^{m-1}} + k. \]

In the above equations, \( k \) and \( y \) are the input and output of AIM, respectively, while \( z_i \) and \( w \) are internal states, and \( B_i(x) \) is the \( \mathbb{F}_2 \)-linearized affine polynomial, i.e.

\[ B_i(x) = a_{i,n} + \sum_{j=0}^{n-1} a_{i,j} x^{2^j}. \]

The corresponding graphic illustration is given in Figure 3.

Three variants of AIM named AIM-I, AIM-II, and AIM-III are specified by the designers, as shown in Table 2.

<table>
<thead>
<tr>
<th>Name</th>
<th>( n )</th>
<th>( m )</th>
<th>( (e_1, \ldots, e_m) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM-I</td>
<td>128</td>
<td>3</td>
<td>(3, 27, 5)</td>
</tr>
<tr>
<td>AIM-II</td>
<td>192</td>
<td>3</td>
<td>(5, 29, 7)</td>
</tr>
<tr>
<td>AIM-III</td>
<td>256</td>
<td>4</td>
<td>(3, 7, 53, 5)</td>
</tr>
</tbody>
</table>

5.2 Low-degree Representation for AIM

We describe the low-degree equivalent representation for AIM in this section. As the algebraic degree of the equivalent representation is relatively high, the polynomial method cannot work efficiently. However, the fast exhaustive search [BCC+10] is still applicable and breaking AIM-I/AIM-II/AIM-III requires about \( 2^{115}/2^{178}/2^{241} \) equivalent calls to the corresponding primitives, respectively.

Specifically, we observe that in AIM, \( m - 1 \) elements in the set \( \{e_1, \ldots, e_m\} \) take small values, i.e. smaller than 8. In what follows, we show how to exploit this property to construct the low-degree representation for AIM.
Given the output \( y \), we can represent \( k \) in terms of the unknown \( w \) as follows:

\[
k = w^{2^m - 1} + y.
\]

In this way, we can further represent each \((z_i)_{1 \leq i \leq m - 1}\) only in terms of \( w \), as shown below:

\[
z_i = (w^{2^m - 1} + y)^{2^{i-1}}.
\]

For convenience, we assume \( e_1 < e_2 < \cdots < e_{m-1} \).

We first show how to compute the accurate algebraic degree for the polynomials \((z_i)_{1 \leq i \leq m - 2}\) in terms of \( w \). Note that

\[
2^i - 1 = \sum_{j=0}^{i-1} 2^j
\]

and hence we have

\[
\forall a, b \in \mathbb{F}_{2^n}, \forall i \in [1, n] : (a + b)^{2^i - 1} = \prod_{j=0}^{i-1} (a + b)^{2^j} = \prod_{j=0}^{i-1} (a^{2^j} + b^{2^j}) = \sum_{j=0}^{2^i-1} a^j b^{2^{i-j}}.
\]

Therefore, we have

\[
z_i = (w^{2^m - 1} + y)^{2^{i-1}} = \sum_{j=0}^{2^i-1} y^j w^{2^{m-1}(2^{i-1} - j)}.
\]

In this way, the algebraic degree of \( z_i \) in terms of \( w \) is

\[
d_i = \max \left\{ \text{Hw} \left( \mathcal{M}_n \left( 2^{m-1} \left( 2^{e_i} - 1 - j \right) \right) \right) \mid j \in [0, 2^{e_i} - 1] \right\},
\]

where

\[
\forall a \in \mathbb{N} : \mathcal{M}_n(a) := \begin{cases} 2^n - 1 & \text{if } 2^n - 1 | a \text{ and } a \geq 2^n - 1, \\ a \% (2^n - 1) & \text{otherwise.} \end{cases}
\]

and Hw\( (a) \) is the hamming weight of \( a \), i.e. the number of 1 in its binary representation. Therefore, \( d_i \) can be naively computed in time \( \mathcal{O}(2^{e_i}) \). Note that for \((e_i)_{1 \leq i \leq m-2}\) in AIM, all of them are smaller than 8 and hence \((d_i)_{1 \leq i \leq m-2}\) can be computed in \( \mathcal{O}(2^8) \).

After computing \((d_i)_{1 \leq i \leq m-2}\), we define

\[
d_{\max} = \max \{d_1, \ldots, d_{m-2}\}.
\]

In this way, \( z_{m-1} \) can be expressed in \( w \) of algebraic degree \( d_{\max} \) due to

\[
z_{m-1} = B_{m-1}^{-1} \left( c + w + \sum_{i=0}^{m-2} B_i (z_i) \right)
\]

\[
= B_{m-1}^{-1} \left( c + w + \sum_{i=0}^{m-2} B_i \left( (w^{2^m - 1} + y)^{2^{e_i}-1} \right) \right)
\]

In other words, the algebraic degree of the above polynomial of \( z_{m-1} \) in terms of \( w \) is \( d_{\max} \).

Furthermore, there is another way to establish the relation between \( z_{m-1} \) and \( w \):

\[
z_{m-1} = (w^{2^m - 1} + y)^{2^{m-1}-1},
\]

13
\( z_{m-1}(w^{2^m-1} + y) = (w^{2^m-1} + y)^{2^m-1} \).

Hence, we obtain an equation only in terms of \( w \), as shown below:

\[
B_{m-1}^{-1}\left(c + w + \sum_{i=0}^{m-2} B_i\left((w^{2^m-1} + y)^{2^i-1}\right)\right)(w^{2^m-1} + y) = (w^{2^m-1} + y)^{2^m-1}
\] (12)

This equation also corresponds to \( n \) Boolean equations of algebraic degree upper bounded by \( d_{\text{max}} + e_m \). These \( n \) Boolean equations can be obtained by polynomial interpolation with the recursive version of Möbius transform, e.g. as described in [Din21a]. To interpolate these \( n \) Boolean polynomials, we need to evaluate

\[
B_{m-1}^{-1}\left(c + w + \sum_{i=0}^{m-2} B_i\left((w^{2^m-1} + y)^{2^i-1}\right)\right)(w^{2^m-1} + y) + (w^{2^m-1} + y)^{2^m-1}
\]

for

\[
T_{\text{num}} = \left( \leq \frac{n}{d_{\text{max}} + e_m} \right)
\] (13)
different \( w \), whose cost can be roughly estimated as \( T_{\text{num}} \) calls to AIM. After the polynomial evaluation is over, recovering the \( n \) Boolean polynomials with these \( T_{\text{num}} \) points require

\[
n \cdot n \cdot \left( \leq \frac{n}{d_{\text{max}} + e_m} \right)
\]

bit operations.

By solving these \( n \) Boolean equations, we can first recover \( w \) and then the secret key can be trivially recovered via:

\[
k = w^{2^m-1} + y.
\]

Remark 1. The designers are indeed aware of this representation, but they use a lower bound on the algebraic degree of \( z_i \) in terms of \( w \), i.e. it is simply treated as the hamming weight of \( M_n\left(2^m - 1\right)2^{e_i-1} \) because the monomial \( x^{(2^m - 1)}2^{e_i-1} \) will appear if we expand the expression [KHS+22]. Moreover, they only treat the polynomial method as a main threat for this low-degree representation, which cannot beat the naive exhaustive search because its memory complexity is much higher than \( 2^n \) bits.

### 5.3 Solving the \( n \) Boolean Equations of Algebraic Degree \( d_{\text{max}} + e_m \)

With the memory-efficient Möbius transform [Din21a,Bou22], evaluating a Boolean polynomial in \( n \) variables of algebraic degree \( d \) over \( \{0, 1\}^n \) requires about \( n \cdot \binom{n}{d} \) bits of memory and the time is about \( d \cdot 2^n \) bit operations. With this as the oracle of the fast exhaustive search method [BCC+10] to efficiently evaluate a polynomial, the time complexity to find the solution of \( w \) from \( n \) Boolean equations of algebraic degree upper bounded by \( d_{\text{max}} + e_m \) is estimated as about

\[
4 \cdot (d_{\text{max}} + e_m) \cdot \log_2 n \cdot 2^n
\] (14)

bit operations. The memory complexity is upper bounded by

\[
n \cdot n \cdot \left( \leq d_{\text{max}} + e_m \right)
\]

bits.
Remark 2. We avoid the original fast exhaustive search method for its costly pre-processing phase of time complexity $O(n^{2(d_{\text{max}} + e_m)})$. However, as already stated in Sect. 2, the time complexity of the pre-processing phase for $n$ Boolean polynomials is indeed upper bounded by

$$n \cdot n^{d_{\text{max}} + e_m} \leq d_{\text{max}} + e_m$$  \hspace{1cm} (15)$$

bit operations. In this sense, using the original fast exhaustive search attack indeed does not affect the whole time complexity as the cost of the pre-processing phase specified in Equation 15 is still much smaller than Equation 14 for the concrete values of $(n, d_{\text{max}} + e_m)$ shown in Table 3.

Our results for the parameters of AIM are summarized in Table 3. Note that due to the relatively small value of $d_{\text{max}} + e_m$ compared with $n$, the cost to interpolate the $n$ Boolean polynomials is negligible, which is almost equivalent to

$$T_{\text{num}} = \left( \frac{n}{d_{\text{max}} + e_m} \right)$$
calls to AIM. Moreover, according to designers’ analysis [KHS+22], without this low-degree equivalent representation in $n$ variables, the naive brute force takes about $2^{149}$, $2^{214.4}$ and $2^{280}$ bit operations for AIM-I, AIM-II and AIM-III, respectively. Hence, the fast exhaustive search attack much improves the naive brute force attack. In other words, the time complexity of our attacks is equivalent to about $2^{115}/2^{178}/2^{241}$ calls of the underlying primitives.

Table 3: Fast exhaustive search (FES) attacks on AIM

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>$n$</th>
<th>$m$</th>
<th>$(e_1, \ldots, e_m)$</th>
<th>$d_{\text{max}} + e_m$</th>
<th>$T_{\text{num}}$</th>
<th>Time (bits)</th>
<th>Memory (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force [KHS+22] FES</td>
<td>128</td>
<td>3</td>
<td>(3, 27, 5)</td>
<td>10</td>
<td>$2^{47.9}$</td>
<td>$2^{149}$</td>
<td>$2^{61.7}$</td>
</tr>
<tr>
<td>Brute force [KHS+22] FES</td>
<td>192</td>
<td>3</td>
<td>(5, 29, 7)</td>
<td>14</td>
<td>$2^{69.3}$</td>
<td>$2^{214.4}$</td>
<td>$2^{84.3}$</td>
</tr>
<tr>
<td>Brute force [KHS+22] FES</td>
<td>256</td>
<td>4</td>
<td>(3, 7, 53, 5)</td>
<td>15</td>
<td>$2^{79.3}$</td>
<td>$2^{280}$</td>
<td>$2^{95.1}$</td>
</tr>
</tbody>
</table>

6 Conclusion

We show that there are nontrivial low-degree equivalent representations for 2-round RAIN and full-round AIM, which can be exploited to mount effective attacks. Especially, as recovering the secret key of AIM is the underlying difficult problem of the signature scheme AIMer, which is one of the NIST Round 1 Additional Signatures, we believe that this work is meaningful to the ongoing NIST PQC competition. While our analysis shows that 3 rounds are sufficient, we propose to primarily use 4-round RAIN with an additional round of security margin.

References


