

# New Low-Memory Algebraic Attacks on LowMC in the Picnic Setting

Fukang Liu<sup>1</sup>, Willi Meier<sup>4</sup>, Santanu Sarkar<sup>5</sup>, Takanori Isobe<sup>1,2,3</sup>

<sup>1</sup> University of Hyogo, Hyogo, Japan

[liufukangs@gmail.com](mailto:liufukangs@gmail.com), [takanori.isobe@ai.u-hyogo.ac.jp](mailto:takanori.isobe@ai.u-hyogo.ac.jp)

<sup>2</sup> NICT, Tokyo, Japan

<sup>3</sup> PRESTO, Japan Science and Technology Agency, Tokyo, Japan

<sup>4</sup> FHNW, Windisch, Switzerland

[willimeier48@gmail.com](mailto:willimeier48@gmail.com)

<sup>5</sup> Indian Institute of Technology Madras, Chennai, India

[santanu@iitm.ac.in](mailto:santanu@iitm.ac.in)

**Abstract.** The security of the post-quantum signature scheme Picnic is highly related to the difficulty of recovering the secret key of LowMC from a single plaintext-ciphertext pair. Since Picnic is one of the alternate third-round candidates in NIST post-quantum cryptography standardization process, it has become urgent and important to evaluate the security of LowMC in the Picnic setting. The best attacks on LowMC with full S-box layers used in Picnic3 were achieved with Dinur’s algorithm. For LowMC with partial nonlinear layers, e.g. 10 S-boxes per round adopted in Picnic2, the best attacks on LowMC were published by Banik et al. with the meet-in-the-middle (MITM) method.

In this paper, we improve the attacks on LowMC in a model where memory consumption is costly. First, a new attack on 3-round LowMC with full S-box layers with negligible memory complexity is found, which can outperform Bouillaguet et al.’s fast exhaustive search attack and can achieve better time-memory tradeoffs than Dinur’s algorithm. Second, we extend the 3-round attack to 4 rounds to significantly reduce the memory complexity of Dinur’s algorithm at the sacrifice of a small factor of time complexity. For LowMC instances with 1 S-box per round, our attacks are shown to be much faster than the MITM attacks. For LowMC instances with 10 S-boxes per round, we can reduce the memory complexity from 32GB ( $2^{38}$  bits) to only 256KB ( $2^{21}$  bits) using our new algebraic attacks rather than the MITM attacks, while the time complexity of our attacks is about  $2^{3.2} \sim 2^5$  times higher than that of the MITM attacks. A notable feature of our new attacks (apart from the 4-round attack) is their simplicity. Specifically, only some basic linear algebra is required to understand them and they can be easily implemented.

**Keywords:** LowMC · Picnic · polynomial method · algebraic attack · crossbred algorithm · low memory

## 1 Introduction

The LowMC block cipher [ARS<sup>+</sup>15] is the first dedicated symmetric-key primitive designed for MPC/FHE/ZK protocols. An important application of LowMC is the Picnic signature scheme [CDG<sup>+</sup>17, KZ20], which is one of the alternate third-round candidates in NIST post-quantum cryptography (PQC) standardization process. Especially, the security of Picnic is directly related to the difficulty of recovering the secret key of LowMC from a single known plaintext-ciphertext pair.

Due to its novel design strategy and its importance, the LowMC block cipher has attracted lots of attention since its publication at EUROCRYPT 2015. However, the security of this novel design is not well-understood and the secure number of rounds of LowMC has been updated several times to resist state-of-the-art attacks [DEM15,DLMW15,RST18]. The latest version is called LowMC v3 and we simply use LowMC to refer to LowMC v3 in the following. Although many powerful attack vectors have been taken into account to determine the secure parameters of LowMC, the recent progress [LIM21,LWM<sup>+</sup>22,Din21] in the cryptanalysis of LowMC indicates that some latest parameters are still insecure. However, the attacks described in [DEM15,DLMW15,RST18,LIM21,LWM<sup>+</sup>22] cannot directly threaten the security of Picnic because the required data complexity is larger than 1.

To understand the security of LowMC in the Picnic setting, the LowMC team launched a public competition<sup>1</sup> in 2019 and some major progress has been made since then. The first important step was made by Banik et al. who found an efficient method [BBDV20] to linearize the 3-bit S-box used in LowMC. Up until now, the best attacks on LowMC with full S-box layers were achieved by Dinur’s algorithm from [Din21] applying the polynomial method [LPT<sup>+</sup>17], which is an advanced method to solve multivariate equation systems over GF(2). However, this technique always requires huge memory and it currently cannot exploit the feature of the well-overdefined equation system. For LowMC with partial nonlinear layers, the best attacks on LowMC were obtained by Banik et al. with the MITM technique [BBVY21]. We notice that the memory complexity of the MITM attacks on LowMC with 10 S-boxes per round is still somewhat high, i.e.  $2^{38}$  bits. Moreover, we observe that the guessed information of the key bits is not exploited at the phase to compute the full key in the MITM attacks. Hence, we are motivated to devise new low-memory attacks on LowMC with a single known plaintext-ciphertext pair.

**Our contributions.** The contributions of this paper are summarized below:

1. We observe that Banik et al.’s guess strategy [BBDV20] to linearize the LowMC S-box is not always optimal for LowMC with full S-box layers. Specifically, to linearize one LowMC S-box, instead of guessing 1 quadratic polynomial in its input, naively guessing 2 linear polynomials in its input can significantly improve the attacks. This is counter-intuitive because guessing 2 bits for each S-box is very costly to linearize 1 round of LowMC compared with guessing only 1 bit.
2. By utilizing a simple version of the crossbred algorithm [JV17] proposed in [BDT22] to solve an overdefined quadratic equation system, our attacks on 3-round LowMC require negligible memory and can achieve better time-memory tradeoffs than Dinur’s algorithm [Din21].
3. To show the effectiveness of our new guess strategy for LowMC with full S-box layers, we also describe attacks on 4-round (full-round) LowMC building upon Dinur’s algorithm. Specifically, we can utilize Dinur’s critical observation [Din21] on the attacks on an odd number of rounds of LowMC to improve the generic complexity of the Dinur’s algorithm. In this way, better time-memory tradeoffs are again achieved, as detailed at the end of Section 3.
4. For Banik et al.’s guess strategy to linearize the S-box [BBDV20], we describe how to construct a well-overdefined system of quadratic equations in key bits to efficiently recover the full key. The new method is especially efficient for LowMC with partial nonlinear layers. The major difference in our new attacks is that we make full use of the guessed quadratic polynomials in key bits while these are not used in Banik et

---

<sup>1</sup><https://lowmcchallenge.github.io/>

al.’s attacks [BBDV20,BBKY21] to recover the key. It can be found that our attacks are much faster than the MITM attacks [BBKY21] for some LowMC parameters.

Our new results for LowMC are shown in Table 1 and Table 2. Note that for attacks on LowMC with partial nonlinear layers, the fast exhaustive search attack [BCC<sup>+</sup>10] cannot work because constructing the polynomial equations is too costly for their high degree. Moreover, the MITM attacks [BBKY21] on 3-round LowMC are slower than the fast exhaustive search attack [BCC<sup>+</sup>10].

**Organization.** In Section 2, we describe the LowMC cipher, introduce how to efficiently evaluate a polynomial for all possible assignments to the variables, and explain a simple version of the crossbred algorithm and Dinur’s algorithm. In Section 3 and Section 4, we demonstrate the new low-memory attacks on LowMC with full S-box layers and with partial nonlinear layers, respectively. Then, in Section 5, we provide the details of our experiments to verify our attacks. Finally, the paper is concluded in Section 6.

**Table 1:** Summary of the best attacks on LowMC with full S-box layers in the Picnic setting, where the time complexity is estimated in bit operations and the memory complexity is estimated in bits.

Methods	$n$	$k$	$s$	$r$	Time	Memory
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{134.8}$	$2^{21}$
Dinur’s algorithm [Din21]	129	129	43	3	$2^{125}$	$2^{104}$
Section 3.2					$2^{127.2}$	$2^{16.9}$
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{197.9}$	$2^{22.7}$
Dinur’s algorithm [Din21]	192	192	64	3	$2^{180}$	$2^{150}$
Section 3.2					$2^{186.2}$	$2^{18.6}$
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{261}$	$2^{24}$
Dinur’s algorithm [Din21]	255	255	85	3	$2^{235}$	$2^{197}$
Section 3.2					$2^{246.8}$	$2^{19.8}$
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{134.8}$	$2^{21}$
Dinur’s algorithm [Din21]	129	129	43	4	$2^{130}$	$2^{113}$
Section 3.3					$2^{133.8}$	$2^{36.7}$
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{197.9}$	$2^{22.7}$
Dinur’s algorithm [Din21]	192	192	64	4	$2^{188}$	$2^{164}$
Section 3.3					$2^{195.0}$	$2^{53.4}$
Fast exhaustive search [BCC <sup>+</sup> 10]					$2^{261}$	$2^{24}$
Dinur’s algorithm [Din21]	255	255	85	4	$2^{245}$	$2^{218}$
Section 3.3					$2^{255.8}$	$2^{68.0}$

## 2 Preliminaries

### 2.1 Description of LowMC

LowMC allows users to choose parameters in a flexible way. Specifically, depending on different scenarios, the users can choose different numbers of rounds, block sizes, key sizes, round constants, linear layers, key schedule functions and the number of S-boxes per round. For convenience, we denote the number of rounds, block size, key size and the number of S-boxes per round by  $r$ ,  $n$ ,  $k$  and  $s$ , respectively. The round function of LowMC is formally

**Table 2:** Summary of the best attacks for  $r = \lfloor \frac{n}{s} \rfloor$  rounds of LowMC with  $s$  S-boxes per round in the Picnic setting, where we estimate one call of  $r$  rounds of LowMC encryption as  $2rn^2$  bit operations as in [BBVY21]

Methods	$n$	$k$	$s$	$r$	Time (#bit operations)	Time (#calls)	Memory (in bits)
MITM [BBVY21] Section 4	128	128	1	128	$2^{147}$ $2^{142.3}$	$2^{125}$ $2^{120.3}$	$2^{22}$ $2^{18.9}$
MITM [BBVY21] Section 4	192	192	1	192	$2^{212.8}$ $2^{205.8}$	$2^{189}$ $2^{182.1}$	$2^{22}$ $2^{19.9}$
MITM [BBVY21] Section 4	256	256	1	256	$2^{278}$ $2^{268.7}$	$2^{253}$ $2^{243.7}$	$2^{22}$ $2^{20.5}$
MITM [BBVY21] Our attack	128	128	10	12	$2^{129.6}$ $2^{134.6}$	$2^{111}$ $2^{116.0}$	$2^{38}$ $2^{18.8}$
MITM [BBVY21] Section 4	192	192	10	19	$2^{199.4}$ $2^{203.7}$	$2^{179}$ $2^{183.2}$	$2^{38}$ $2^{20.0}$
MITM [BBVY21] Section 4	256	256	10	25	$2^{259.6}$ $2^{262.8}$	$2^{238}$ $2^{241.2}$	$2^{38}$ $2^{20.6}$

described below:

$$A^{i+1} = L_i \cdot \mathcal{S}(A^i) \oplus RC_i \oplus M_i \cdot K,$$

where  $A^i$  is the input state of the  $i$ -th round,  $K$  is the  $k$ -bit key,  $RC_i$  is the used  $n$ -bit round constant,  $M_i$  is a full-rank matrix of size  $n \times k$ ,  $L_i$  is a full-rank matrix of size  $n \times n$  and  $\mathcal{S}$  is the nonlinear operation by applying  $s$  3-bit S-boxes to the first  $3s$  bits of  $A^i$  in parallel.

Denote the plaintext and ciphertext by  $p$  and  $c$ , respectively. Then, we have

$$A^1 = p \oplus M_0 \cdot K, \quad c = A^{r+1},$$

where  $M_0$  is also a full rank matrix of size  $n \times k$ . Note that  $M_0 \cdot K$  is the whitening key.

For LowMC,  $RC_i$ ,  $L_i$  and  $M_j$  ( $1 \leq i \leq r, 0 \leq j \leq r$ ) are all randomly generated, i.e. they are not fixed as in many other block ciphers. Specifically, before encrypting  $p$ , we need to compute the parameters for  $(RC_i, L_i, M_j)$  with a pseudorandom number generator.

The 3-bit S-box  $S(a_1, a_2, a_3) = (a_4, a_5, a_6)$  of LowMC is defined as below:

$$a_4 = a_1 \oplus a_2 a_3, \quad a_5 = a_1 \oplus a_2 \oplus a_1 a_3, \quad a_6 = a_1 \oplus a_2 \oplus a_3 \oplus a_1 a_2,$$

where  $(a_1, a_2, a_3) \in \mathbb{F}_2^3$  and  $(a_4, a_5, a_6) \in \mathbb{F}_2^3$  denote the input and output, respectively. Note that the inverse of this S-box can be easily deduced, as shown below:

$$a_1 = a_4 \oplus a_5 \oplus a_5 a_6, \quad a_2 = a_5 \oplus a_4 a_6, \quad a_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_4 a_5.$$

## 2.2 Evaluating a Polynomial of Degree $d$ in $u$ Variables

Given a polynomial  $f(x) \in \mathbb{F}_2[x_1, x_2, \dots, x_u]$  of degree  $d$ , we aim to evaluate  $f(x)$  over all  $x = (x_1, x_2, \dots, x_u) \in \mathbb{F}_2^u$ . A naive algorithm is to evaluate each term of  $f(x)$  for each assignment to  $x$ , which results in a time complexity upper bounded by about  $2^u \cdot \binom{u}{\leq d}$  bit operations, where  $\binom{u}{\leq d} = \sum_{i=0}^d \binom{u}{i}$ . By simply utilizing the Möbius transform, it is possible to achieve this purpose with  $u \cdot 2^u$  bit operations. For simplicity, we also write a function  $f(x)$  as  $f$  in the following.

To see how it works, it is necessary to realize that the Möbius transform is an involution on the set of Boolean functions. For any such  $f$ , we can write its algebraic normal form (ANF) as follows:

$$f(x) = \bigoplus_{b=(b_1, b_2, \dots, b_u) \in \mathbb{F}_2^u} g(b_1, b_2, \dots, b_u) \prod_{i=1}^u x_i^{b_i},$$

where  $g : \mathbb{F}_2^u \rightarrow \mathbb{F}_2$  is a Boolean function and  $g(b_1, b_2, \dots, b_u)$  is the coefficient of the term  $\prod_{i=1}^u x_i^{b_i}$ .

Recall the common usage of the Möbius transform. Given a blackbox access to the polynomial  $f$ , we can construct the truth table denoted by `TAB_F` of  $(x, f)$  by exhausting  $2^u$  possible values of  $x$ . Based on this truth table, we can then determine the accurate ANF of  $f$  with  $u \cdot 2^u$  bit operations, i.e. we obtain a truth table denoted by `TAB_B` of  $(b, g)$ . Indeed, the Möbius transform is just to apply a butterfly-style algorithm to the table `TAB_F` to obtain a new table `TAB_B`. If we again apply the Möbius transform to the table `TAB_B`, we will obtain the table `TAB_F`, which is why we say it is an involution.

To see the reasons, let us first focus on the relation between  $g(b)$  and  $f(x)$ . For any  $b$ , we introduce a set of indices  $J = \{i_1, i_2, \dots, i_j\} \subseteq \{1, 2, \dots, u\}$  such that  $b_i = 1$  ( $i \in J$ ) and  $b_{i'} = 0$  ( $i' \notin J$ ). Then, we have

$$g(b) = \bigoplus_{(x_{i_1}, x_{i_2}, \dots, x_{i_j}) \in \mathbb{F}_2^j, x_{i'} \leftarrow 0} f(x). \quad (1)$$

The Möbius transform is indeed to compute  $g(b)$  for all  $b \in \mathbb{F}_2^u$  with the above formula based on `TAB_F` in only  $u \cdot 2^u$  bit operations.

Indeed, based on the ANF of  $f$ , for any  $x$  with  $x_i = 1$  ( $i \in J$ ) and  $x_{i'} = 0$  ( $i' \notin J$ ), we also have

$$f(x) = \bigoplus_{(b_{i_1}, b_{i_2}, \dots, b_{i_j}) \in \mathbb{F}_2^j, b_{i'} \leftarrow 0} g(b). \quad (2)$$

Therefore, if we apply the Möbius transform to the table `TAB_B`, we will obtain the table `TAB_F`.

In a word, given any concrete  $f$ , we can evaluate  $f$  over all  $x \in \mathbb{F}_2^u$  in  $u \cdot 2^u$  bit operations and with  $2^u$  memory in bits because the truth table of  $(b, g)$  can be directly extracted from the expression of  $f$ . With some extra efforts in the procedure to do the Möbius transform, the time complexity can be reduced to  $d \cdot 2^u$ , as shown in [DS11].

In Dinur's algorithm [Din21], the memory complexity of the above standard Möbius transform is further reduced to about  $u \cdot \binom{u}{\leq d}$  bits and the time complexity is almost kept the same, i.e.  $d \cdot 2^u$  bit operations.

**Using Gray code for the case  $d = 2$ .** For most of our new attacks on LowMC, we need to handle the case  $d = 2$ . Therefore, we use a very simple yet efficient method [BCC<sup>+</sup>10, BDT22] to evaluate a quadratic polynomial based on *Gray code*. The idea is based on the fact that when only one variable changes, the values of at most  $u$  terms in  $f$  will change.

When  $d = 2$ ,  $f$  can be represented as

$$f = c \oplus \bigoplus_{i=1}^u \bigoplus_{j=i}^u a_{i,j} x_i x_j,$$

where  $c, a_{i,j} \in \mathbb{F}_2$ . Let  $e_i \in \mathbb{F}_2^u$  denote the unit vector which is zero everywhere except on the  $i$ -th coordinate. Then, we have

$$f(x) \oplus f(x \oplus e_i) = a_{i,i} \oplus \bigoplus_{j=1}^{i-1} a_{j,i} x_j \oplus \bigoplus_{j=i+1}^u a_{i,j} x_j.$$

Hence, we can enumerate all the  $2^u$  values of  $x$  using Gray code so that only a single bit of  $x$  changes at each iteration. In this way, evaluating  $f$  over  $x \in \mathbb{F}_2^u$  only takes  $u \cdot 2^u$  bit operations and the memory complexity is negligible. It is now easy to observe that when  $d = 1$ , evaluating  $f$  over  $x \in \mathbb{F}_2^u$  with Gray code only takes  $2^u$  bit operations.

## 2.3 A Simple Version of Crossbred Algorithm for Quadratic Equations

In our attacks, we will adopt a simple version of the crossbred algorithm [JV17] to solve an overdefined system of quadratic equations, which is described in [BDT22]. This algorithm fits very well with our attacks on LowMC for its simplicity to bound the time complexity and to implement in practice.

Consider a system of  $m$  quadratic Boolean equations  $f_t(x) = 0$  ( $1 \leq t \leq m$ ) in  $u$  variables  $x = (x_1, x_2, \dots, x_u) \in \mathbb{F}_2^u$ . First, split these  $u$  variables into two parts  $y \in \mathbb{F}_2^{u-u_1}$  and  $z \in \mathbb{F}_2^{u_1}$ , where  $y = (y_1, y_2, \dots, y_{u-u_1}) = (x_{u_1+1}, x_{u_1+2}, \dots, x_u)$  and  $z = (z_1, z_2, \dots, z_{u_1}) = (x_1, x_2, \dots, x_{u_1})$ . Then, we write each quadratic polynomial  $f_t$  ( $1 \leq t \leq m$ ) as follows:

$$f_t = \bigoplus_{i=1}^{u_1} \bigoplus_{j=i+1}^{u_1} \alpha_{i,j} z_i z_j \oplus \bigoplus_{i=1}^{u_1} l_i(y) z_i \oplus q_t(y),$$

where  $\alpha_{i,j} \in \mathbb{F}_2$  and both  $l_i(y)$  and  $q_t(y)$  are polynomials in  $y$  with  $\text{Deg}(l_i) = 1$  and  $\text{Deg}(q_t) = 2$ . Throughout this paper, we denote the degree of a polynomial  $f$  by  $\text{Deg}(f)$ .

For example, consider the case when  $(u, u_1) = (6, 3)$  and

$$f_t = z_1 z_2 \oplus z_1 z_3 \oplus y_1 z_2 \oplus y_2 z_2 \oplus y_1 z_3 \oplus y_1 y_2 \oplus y_3.$$

We can rewrite the above  $f_t$  as

$$f_t = (1, 1, 0, 0, y_1 \oplus y_2, y_1) \cdot (z_1 z_2, z_1 z_3, z_2 z_3, z_1, z_2, z_3)^T \oplus (y_1 y_2 \oplus y_3),$$

where

$$(\alpha_{1,2}, \alpha_{1,3}, \alpha_{2,3}) = (1, 1, 0), (l_1, l_2, l_3) = (0, y_1 \oplus y_2, y_1), q_t = y_1 y_2 \oplus y_3.$$

In other words, the system of  $m$  quadratic equations can be expressed as follows:

$$A \cdot (z_1 z_2, z_1 z_3, \dots, z_{u_1-1} z_{u_1}, z_1, z_2, \dots, z_{u_1})^T = B, \quad (3)$$

where  $A$  is the coefficient matrix of size  $m \times (u_1(u_1 - 1)/2 + u_1)$  and  $B$  is a vector of size  $m$ . Moreover, for the elements in the first  $u_1(u_1 - 1)/2$  columns of  $A$ , all of them take constant values from  $\{0, 1\}$  according to  $f_t$  ( $1 \leq t \leq m$ ). For the elements in last  $u_1$  columns of  $A$ , each of them is written as a linear polynomial in  $y$ , i.e. the coefficient of  $z_i$  ( $1 \leq i \leq u_1$ ) in each  $f_t$  ( $1 \leq t \leq m$ ) is a linear polynomial in  $y$ . For the vector  $B$ , each element is written as a quadratic polynomial in  $y$ .

For example, consider the following system of quadratic equations in  $(z_1, z_2, z_3, y_1, y_2, y_3)$ :

$$\begin{aligned} f_1 &= z_1 z_2 \oplus z_1 z_3 \oplus y_1 z_2 \oplus y_2 z_2 \oplus y_1 z_3 \oplus y_1 y_2 \oplus y_3 = 0, \\ f_2 &= z_1 z_3 \oplus z_2 z_3 \oplus y_1 z_1 \oplus y_2 z_2 \oplus y_2 z_3 \oplus y_1 y_3 \oplus y_3 = 0, \\ f_3 &= z_1 z_2 \oplus y_1 y_2 \oplus y_2 y_3 \oplus y_3 \oplus 1 = 0. \end{aligned}$$

Then, we can rewrite it in the form shown in Equation 3, as specified below:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & y_1 \oplus y_2 & y_1 \\ 0 & 1 & 1 & y_1 & y_2 & y_2 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} z_1 z_2 \\ z_1 z_3 \\ z_2 z_3 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} y_1 y_2 \oplus y_3 \\ y_1 y_3 \oplus y_3 \\ y_1 y_2 \oplus y_2 y_3 \oplus y_3 \oplus 1 \end{bmatrix} \quad (4)$$

With the representation described in Equation 3 in mind, it is easy to understand the simple algorithm to solve  $m$  quadratic equations in  $u$  variables. The overall procedure can be described as follows:

1. Apply the Gaussian elimination on the augmented matrix  $A|B$  such that the first  $u_1(u_1 - 1)/2$  columns of the matrix  $A|B$  are in the reduced row echelon form. Denote the matrix after this Gaussian elimination operation by  $A'|B'$ .
2. From the last  $m - u_1(u_1 - 1)/2$  rows of the matrix  $A'|B'$ , we can deduce  $m - u_1(u_1 - 1)/2$  equations of the form

$$A'' \cdot (z_1, z_2, \dots, z_{u_1})^T = B'', \quad (5)$$

where each element in  $A''$  is a linear polynomial in  $y$  and each element in  $B''$  is a quadratic polynomial in  $y$ .

3. Using Gray code to exhaust all the  $2^{u-u_1}$  possible values of  $y$ . For each value of  $y$ , update  $A''$  and  $B''$  and solve the linear equation system Equation 5 using Gaussian elimination. If there is a solution, check its correctness against the original  $m$  equations<sup>2</sup>. If there is no solution, try another value of  $y$ .

**Complexity analysis.** To efficiently solve the quadratic equations with the above method, it is required that the equation system Equation 5 is slightly overdefined, i.e. for each guess of  $y$ , there is at most one solution of  $z$ . Another reason to make it slightly overdefined is to amortize the cost to check the solutions. Let

$$\epsilon + u_1 = m - u_1(u_1 - 1)/2, \epsilon > 0.$$

The time complexity of Step 1 – 2 can be estimated as

$$m^2 \cdot \binom{u}{\leq 2}$$

bit operations as we need to perform the addition operation for quadratic polynomials in  $u$  variables for each row operation.

The time complexity of Step 3 can be estimated as the sum of the time complexity to update  $(A'', B'')$ , the time complexity to solve Equation 5 and the time complexity to check the original  $m$  equations. We assume that Equation 5 is slightly overdefined and the cost to check the correctness of the solutions is negligible<sup>3</sup>. Then, the time complexity of Step 3 can be estimated as

$$(\epsilon + u_1)((u - u_1) + u_1) \cdot 2^{u-u_1} + (\epsilon + u_1)^2 \cdot u_1 \cdot 2^{u-u_1} = 2^{u-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + u)$$

bit operations. Therefore, the total time complexity is estimated as

$$m^2 \cdot \binom{u}{\leq 2} + 2^{u-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + u)$$

bit operations.

<sup>2</sup>A better way is to choose the quadratic equations from  $A'|B'$  that are not included in  $A''|B''$  for verification.

<sup>3</sup>To check the correctness of a solution, we can randomly pick a quadratic equation not included in  $A''|B''$  to check its correctness. For the wrong solution, we expect each such equation holds with probability  $2^{-1}$ . Thus, we estimate the time complexity to check  $2^\omega$  solutions as  $\binom{u}{\leq 2} \cdot \sum_{i=0}^{\omega} 2^{\omega-i} \approx 2^{\omega+1} \cdot \binom{u}{\leq 2}$  bit operations. The value of  $\omega$  can be estimated as  $\omega = u - u_1 - \epsilon$ , thus resulting in a complexity to check the solutions of  $2^{u-u_1-\epsilon+1} \cdot \binom{u}{\leq 2}$  bit operations. This complexity in most cases will be smaller than  $2^{u-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + u)$ , i.e.  $2^{-\epsilon+1} \cdot \binom{u}{\leq 2} < (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + u)$ . We have checked that this holds for all our attacks.

## 2.4 On Dinur's Algorithm

In our attacks on full-round (4-round) LowMC with full S-box layers, we will use Dinur's algorithm [Din21] to solve a system of nonlinear equations of degree 4, which is based on the polynomial method originally proposed in [LPT<sup>+</sup>17]. Similar to Dinur's attack on LowMC of an odd number of rounds, we can optimize the generic time complexity given in [Din21] for such an equation system by taking the structure of equations into account. Therefore, it is necessary to understand how Dinur's algorithm works and how the complexity is computed. In the following, we will describe a simplified version of Dinur's algorithm.

Similarly, we consider the system of  $m$  equations denoted by  $E(x)$  as shown below:

$$E(x) : P_1(x) = P_2(x) = \dots = P_m(x) = 0, \quad (6)$$

where  $\text{Deg}(P_i) = d$  ( $1 \leq i \leq m$ ) and  $x = (x_1, x_2, \dots, x_u) \in \mathbb{F}_2^u$ . First, split the  $u$  variables into  $y \in \mathbb{F}_2^{u-u_1}$  and  $z \in \mathbb{F}_2^{u_1}$ , where  $y = (y_1, y_2, \dots, y_{u-u_1}) = (x_{u_1+1}, x_{u_1+2}, \dots, x_u)$  and  $z = (z_1, z_2, \dots, z_{u_1}) = (x_1, x_2, \dots, x_{u_1})$ . Then,  $P_i(x)$  can also be written as  $P_i(y, z)$ .

Next, we randomly take  $\ell = u_1 + 1$  different equations from the  $m$  equations  $P_i(y, z) = 0$  and we denote the new system of equations by  $\tilde{E}(y, z)$  as shown below:

$$\tilde{E}(y, z) : R_1(y, z) = R_2(y, z) = \dots = R_\ell(y, z) = 0. \quad (7)$$

It is easy to observe that a solution to Equation 6 must be a solution to Equation 7. However, a solution to Equation 7 is not necessarily a solution to Equation 6. The general idea of Dinur's algorithm is to efficiently enumerate all the solutions to Equation 7 and then test their correctness against Equation 6.

**Assumption.** We assume that when the value of  $y$  is specified, there is at most 1 solution of  $z$  satisfying Equation 7, and the corresponding  $(y, z)$  is called the isolated solution to  $\tilde{E}(y, z)$ .

**Definition 1.** [Din21] If  $\hat{x} = (\hat{y}, \hat{z})$  is a solution to  $\tilde{E}(y, z)$  and for any  $z' \neq \hat{z}$ ,  $(\hat{y}, z')$  cannot be a solution to  $\tilde{E}(y, z)$ , then  $\hat{x} = (\hat{y}, \hat{z})$  is called the **isolated solution** to  $\tilde{E}(y, z)$ .

Let us elaborate more on the isolated solution. According to its definition, for each specified value of  $y$  denoted by  $\hat{y}$ , there is at most 1 solution of  $z$  denoted by  $\hat{z}$  satisfying  $\tilde{E}(y, z)$  and if such a  $\hat{z}$  exists,  $(\hat{y}, \hat{z})$  is an isolated solution to  $\tilde{E}(y, z)$ . Hence, it is indeed equivalent to the assumption made above.

The above assumption is reasonable because once  $y$  is specified, we get  $\ell = u_1 + 1$  equations in  $u_1$  variables. Hence, we can expect that there is at most 1 solution of  $z$ . We emphasize here that we have checked the probability of this assumption in our 4-round attacks on LowMC, as detailed in Section 5.

Now suppose we have a blackbox function denoted by  $U(y) : \mathbb{F}_2^{u-u_1} \rightarrow \mathbb{F}_2^{u_1}$  which can efficiently output the value of  $\hat{z}$  for each given  $\hat{y}$  such that  $(\hat{y}, \hat{z})$  is an isolated solution to  $\tilde{E}(y, z)$ . In this way, the time complexity to enumerate all the solutions to  $\tilde{E}(y, z)$  is identical to the time complexity to evaluate the function  $U(y)$  over  $\mathbb{F}_2^{u-u_1}$ , which can be efficiently finished via Möbius transform once the explicit Algebraic Normal Forms (ANFs) of  $U(y)$  are known.

**Finding the blackbox function  $U(y)$ .** The core idea of Dinur's algorithm is indeed to find the above blackbox function  $U(y)$  with time complexity less than  $2^u$ . To achieve this, let us consider an equivalent representation of the equation system  $\tilde{E}(y, z)$  as below:

$$\tilde{F}(y, z) = (R_1(y, z) \oplus 1)(R_2(y, z) \oplus 1) \dots (R_\ell(y, z) \oplus 1).$$

In this way, the set of solutions to  $\tilde{E}(y, z)$  is identical to the set of solutions to the equation:

$$\tilde{F}(y, z) = 1.$$

This is because  $\tilde{F}(y, z) = 1$  is equivalent to  $R_i(y, z) = 0$  for  $1 \leq i \leq \ell$ .

Since  $\text{Deg}(R_i) = d$ , we have  $d_{\tilde{F}} = \text{Deg}(\tilde{F}) \leq \ell \cdot d$ . Then, we rewrite  $\tilde{F}(y, z)$  in different forms as shown below:

$$\begin{aligned}\tilde{F}(y, z) &= z_1 z_2 \dots z_{u_1} U_0(y) \oplus Q_0(y, z), \\ \tilde{F}(y, z) &= z_1 z_2 \dots z_{i-1} z_{i+1} \dots z_{u_1} U_i(y) \oplus Q_i(y, z) \text{ where } z_i = 0.\end{aligned}$$

Similar to the cube attack [DS09], there is no term in  $Q_0(y, z)$  which is divisible by  $z_1 z_2 \dots z_{u_1}$  and there is no term in  $Q_i(y, z)$  which is divisible by  $z_1 z_2 \dots z_{i-1} z_{i+1} \dots z_{u_1}$ . Moreover, we also have

$$\begin{aligned}U_0(y) &= \bigoplus_{z \in \mathbb{F}_2^{u_1}} \tilde{F}(y, z), \\ U_i(y) &= \bigoplus_{(z_1, z_2, \dots, z_{i-1}, z_{i+1}, \dots, z_{u_1}) \in \mathbb{F}_2^{u_1-1}, z_i=0} \tilde{F}(y, z) \text{ where } 1 \leq i \leq u_1, \\ d_{U_0} &= \text{Deg}(U_0) \leq d_{\tilde{F}} - u_1, \\ d_{U_i} &= \text{Deg}(U_i) \leq d_{\tilde{F}} - u_1 + 1 \text{ where } 1 \leq i \leq u_1.\end{aligned}$$

**Relations between  $U_i(y)$  and the isolated solutions.** Supposing  $(\hat{y}, \hat{z})$  is an isolated solution to  $\tilde{E}(y, z)$ , we can trivially have

$$U_0(\hat{y}) = 1$$

because there is only 1 value  $z = \hat{z}$  which can make  $\tilde{F}(\hat{y}, z) = 1$ . If there is no solution to  $\tilde{E}(y, z)$  for the given  $y = \hat{y}$ , there will be  $U_0(\hat{y}) = 0$  because for each  $z \in \mathbb{F}_2^{u_1}$ ,  $\tilde{F}(\hat{y}, z) = 0$ . Note that we make an assumption that there is either no solution or 1 solution to  $\tilde{E}(y, z)$  for a given  $y$ . Hence,  $U_0(\hat{y})$  can help determine whether there is a solution to  $\tilde{E}(y, z)$  for a given  $y = \hat{y}$ . Once it is determined that there is 1 solution to  $z$ , the remaining work is to deduce this solution  $\hat{z}$  under the given  $y = \hat{y}$ .

To deduce  $\hat{z}$  once  $U_0(\hat{y}) = 1$ , it is necessary to observe that

$$\hat{z}_i = U_i(y) \oplus 1 \text{ where } 1 \leq i \leq u_1.$$

This can be verified and proved under the above mentioned assumption [Din21]. Hence, for a given  $y = \hat{y}$ , computing the solution to  $\tilde{E}(\hat{y}, z)$  can be described as follows:

1. Compute  $U_i(y)$  for all  $0 \leq i \leq u_1$ .
2. If  $U_0(y) = 0$ , there is no solution and exit.
3. If  $U_0(y) = 1$ , make  $\hat{z}_i = U_i(y) \oplus 1$ . Then,  $(\hat{y}, \hat{z})$  is a solution to  $\tilde{E}(y, z)$ .

Therefore, the last task is to recover the explicit formula of  $U_i(y)$  ( $0 \leq i \leq u_1$ ). Note that  $d_{U_0} \leq d_{\tilde{F}} - u_1 = w$  and  $d_{U_i} \leq d_{\tilde{F}} - u_1 + 1 = w + 1$  for  $1 \leq i \leq u_1$ .

**Recovering  $(U_0(y), U_1(y), \dots, U_{u_1}(y))$ .**  $U_0(y)$  can be interpolated from its value set  $W_w^{u-u_1}$ , where  $W_w^{u-u_1} = \{a | HW(a) \leq w\}$  and  $HW(a)$  is the Hamming weight of  $a$ . To compute each value for such a set, we need to do  $2^{u_1}$  evaluations of  $\tilde{F}(y, z)$ . Similarly,  $U_i(y)$  ( $1 \leq i \leq u_1$ ) can be interpolated from its value set  $W_{w+1}^{u-u_1}$ , where the computation of each such value requires  $2^{u_1-1}$  evaluations of  $\tilde{F}(y, z)$ . In other words, to interpolate

$U_0(y)$ , we need a list of  $\tilde{F}(y, z)$  of size  $\binom{u-u_1}{\leq w} \times 2^{u_1}$ , i.e. we need to exhaust the solutions of  $\tilde{E}(y, z)$  from the constrained space  $(y, z) \in W_w^{u-u_1} \times \{0, 1\}^{u_1}$ . Similarly, to interpolate  $U_i(y)$  ( $1 \leq i \leq u_1$ ), we need to exhaust the solutions of  $\tilde{E}(y, z)$  from the constrained space  $(y, z) \in W_{w+1}^{u-u_1} \times \{0, 1\}^{i-1} \times \{0\} \times \{0, 1\}^{u_1-i}$ . Hence, we only need to exhaust all the solutions to  $\tilde{E}(y, z)$  in the constrained space  $(y, z) \in W_{w+1}^{u-u_1} \times \{0, 1\}^{u_1}$ , which is sufficient to compute all the required lists. Exhausting the solutions to  $\tilde{E}(y, z)$  in the constrained space  $(y, z) \in W_{w+1}^{u-u_1} \times \{0, 1\}^{u_1}$  requires

$$2d \cdot \log_2 u \cdot 2^{u_1} \cdot \binom{u-u_1}{\leq w+1} \quad (8)$$

bit operations using the fast exhaustive search algorithm [BCC<sup>+</sup>10].

Note that we only record those  $(y, z)$  such that  $\tilde{F}(y, z) = 1$ , i.e. we only care about those  $(y, z)$  which are solutions to  $\tilde{E}(y, z)$ . In this way, we will obtain a list of  $(y, z)$  of size of about  $\frac{1}{2} \cdot \binom{u-u_1}{\leq w+1}$ . After obtaining the list, we need to construct an array of size  $\binom{u-u_1}{\leq w}$  for  $U_0(y)$  according to its definition. We also need to construct  $u_1$  other arrays of size  $\binom{u-u_1}{\leq w+1}$  for  $U_i(y)$  ( $1 \leq i \leq u_1$ ) according their definitions. The time complexity of this phase is slightly larger than  $(u_1 + 1) \cdot \frac{1}{2} \cdot \binom{u-u_1}{\leq w+1}$  but is still negligible compared with Equation 8. Then, with these arrays, we can interpolate  $(U_0(y), U_1(y), \dots, U_{u_1}(y))$  with the Möbius transform, which requires less than  $(u_1 + 1) \cdot \binom{u-u_1}{\leq w+1}$  bit operations and is negligible compared with Equation 8. Hence, the time complexity to recover  $(U_0(y), U_1(y), \dots, U_{u_1}(y))$  is dominated by Equation 8, i.e. it is

$$2d \cdot \log_2 u \cdot 2^{u_1} \cdot \binom{u-u_1}{\leq w+1} = 2d \cdot \log_2 u \cdot 2^{u_1} \cdot \binom{u-u_1}{\leq d_{\tilde{F}} - u_1 + 1}.$$

We refer the interested readers to [Din21] for more details of this procedure as it is the core of Dinur's algorithm and explaining the full details in such a short paragraph is infeasible.

**The total time complexity.** To reduce the overload to check the correctness of solutions and to efficiently identify the isolated solution, we may prepare four different choices for the equation system  $\tilde{E}(y, z)$  and we perform the above procedure for each of 4 equation systems. Then, the time complexity can be estimated as

$$4 \cdot \left( 2d \cdot \log_2 u \cdot 2^{u_1} \cdot \binom{u-u_1}{\leq d_{\tilde{F}} - u_1 + 1} \right) + 4 \cdot (u_1 + 1) \cdot (u - u_1) \cdot 2^{u-u_1}$$

bit operations, where the second term accounts for evaluating  $u_1 + 1$  polynomials on the space  $\{0, 1\}^{u-u_1}$  for four times.

**The total memory complexity.** At the phase to recover all  $U_i(y)$ , it is required to construct a table of size of about  $4 \cdot \frac{1}{2} \cdot \binom{u-u_1}{\leq d_{\tilde{F}} - u_1 + 1}$  bits so that the Möbius transform can work. At the phase to evaluate  $(u_1 + 1)$  polynomials over the space  $\{0, 1\}^{u-u_1}$ , with the standard Möbius transform, the memory complexity is about  $4 \cdot (u_1 + 1) \cdot 2^{u-u_1}$  bits. If using Dinur's memory-efficient Möbius transform, the memory complexity is about  $4 \cdot (u_1 + 1) \cdot \binom{u-u_1}{\leq d_{\tilde{F}} - u_1 + 1}$ .

### 3 New Algebraic Attacks on LowMC

In this section, we will first demonstrate 3 different attacks on LowMC with full S-box layers. The first attack is a new and simple guess-and-determine (GnD) attack on 3-round LowMC by using Banik et al.'s strategy [BBDV20] to linearize the 3-bit S-box, where we

solve a system of quadratic equations with the standard linearization technique. The second attack is a much simpler yet more efficient GnD attack on 3-round LowMC by using a naive guess strategy to linearize the 3-bit S-box, where we solve quadratic equations with the simplified version of the crossbred algorithm [BDT22]. The third attack is for full-round (4-round) LowMC, where we still adopt the naive guess strategy but use Dinur's algorithm [Din21] to solve equations of degree 4.

### 3.1 The First Attack on LowMC

The first attack is essentially based on the strategy to linearize the 3-bit S-box by guessing a quadratic relation, as first proposed in [BBDV20]. Specifically, consider the definition of the S-box as shown below:

$$a_4 = a_1 \oplus a_2 a_3, \quad a_5 = a_1 \oplus a_2 \oplus a_1 a_3, \quad a_6 = a_1 \oplus a_2 \oplus a_3 \oplus a_1 a_2.$$

If we guess  $a_4 = a_4^*$ , i.e. the value of the quadratic polynomial  $a_1 \oplus a_2 a_3$  is  $a_4^*$ , then  $(a_4, a_5, a_6)$  can be expressed as linear expressions in terms of  $(a_1, a_2, a_3)$ :

$$\begin{aligned} a_4 &= a_4^* \\ a_5 &= a_1 \oplus a_2 \oplus (a_4^* \oplus a_2 a_3) a_3 = a_4^* \oplus a_2 \oplus a_3 a_4^*, \\ a_6 &= a_1 \oplus a_2 \oplus a_3 \oplus (a_4^* \oplus a_2 a_3) a_2 = a_4^* \oplus a_2 \oplus a_3 \oplus a_2 a_4^*, \end{aligned}$$

Indeed, guessing any 1 non-zero linear polynomial in  $(a_4, a_5, a_6)$  can help linearize the 3-bit S-box, i.e.  $(a_4, a_5, a_6)$  can then be expressed as linear expressions in terms of  $(a_1, a_2, a_3)$ . This property also applies to its inverse due to the similar formulas for its inverse:

$$a_1 = a_4 \oplus a_5 \oplus a_5 a_6, \quad a_2 = a_5 \oplus a_4 a_6, \quad a_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_4 a_5.$$

In other words, guessing any 1 non-zero linear polynomial in  $(a_1, a_2, a_3)$  can make  $(a_1, a_2, a_3)$  linear in  $(a_4, a_5, a_6)$  [BBDV20].

**An overdefined system of equations for the 3-bit S-box.** Since the algebraic attack on AES was published [CP02], though it is commonly believed the attack cannot work as expected, it has been well-known that one can find an overdefined system of linearly independent quadratic equations for the used S-box. For the 3-bit S-box of LowMC, we can find at most 14 linearly independent quadratic equations, as shown below:

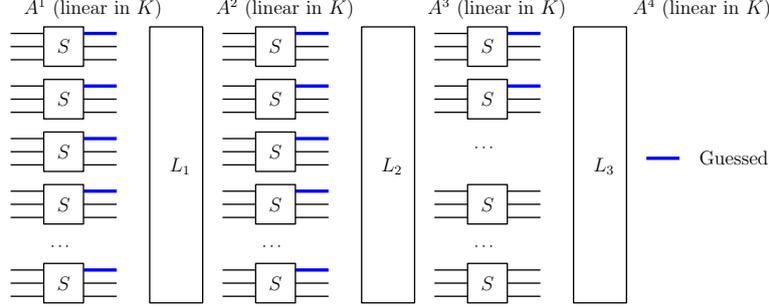
$$\begin{aligned} a_4 &= a_1 \oplus a_2 a_3, \quad a_5 = a_1 \oplus a_2 \oplus a_1 a_3, \quad a_6 = a_1 \oplus a_2 \oplus a_3 \oplus a_1 a_2, \\ a_1 &= a_4 \oplus a_5 \oplus a_5 a_6, \quad a_2 = a_5 \oplus a_4 a_6, \quad a_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_4 a_5, \\ a_4 a_2 &= a_1 a_2 \oplus a_2 a_3, \quad a_4 a_3 = a_1 a_3 \oplus a_2 a_3, \quad a_5 a_1 = a_1 \oplus a_1 a_2 \oplus a_1 a_3, \\ a_5 a_3 &= a_2 a_3, \quad a_6 a_1 = a_1 \oplus a_1 a_3, \quad a_6 a_2 = a_2 \oplus a_2 a_3, \\ a_4 a_1 \oplus a_1 &= a_5 a_2 \oplus a_1 a_2 \oplus a_2, \quad a_5 a_2 \oplus a_1 a_2 \oplus a_2 = a_6 a_3 \oplus a_1 a_3 \oplus a_2 a_3 \oplus a_3. \end{aligned}$$

**The GnD attack by solving quadratic equations.** We noticed that in Banik et al.'s attacks [BBDV20, BBVY21] on LowMC, the guessed quadratic polynomials in key bits are not used to recover the key and they are only used to linearize the S-box. In this attack, we will make full use of all the guessed polynomials and the overdefined system of quadratic equations for the 3-bit S-box. First, we show that at each time we guess a quadratic polynomial for the 3-bit S-box, we indeed can obtain 3 quadratic equations rather than only 1 quadratic equation.

Let us take the guess  $a_4 = a_4^*$  for an instance. In this case, we have the following 3 linearly independent quadratic equations in  $(a_1, a_2, a_3)$ :

$$a_4^* = a_1 \oplus a_2 a_3, \quad a_4^* a_2 = a_1 a_2 \oplus a_2 a_3, \quad a_4^* a_3 = a_1 a_3 \oplus a_2 a_3.$$

Indeed, if we guess any 1 linear equation in  $(a_4, a_5, a_6)$ , we can always obtain 3 quadratic equations in terms of  $(a_1, a_2, a_3)$  from such a guess, which can be easily derived based on a similar strategy.



**Figure 1:** Illustration of the first attack on 3-round LowMC

With the above observations in mind, it is now easy to understand our first GnD attack. As shown in Fig. 1, for each S-box in the first 2 rounds, we linearize it by guessing 1 output bit. In this way, each input state bit of the S-box at the first 3 rounds can be written as linear expressions in the  $n$ -bit key  $K = (K_1, K_2, \dots, K_n)$ . In addition, for such a guess strategy, we can obtain  $3 \cdot 2s = 6s$  quadratic equations in  $K$  because we guess  $2s$  quadratic equations for the  $2s$  S-boxes in the first 2 rounds to linearize them. Finally, we deal with the third round to attack 3-round LowMC.

For the  $s$  S-boxes in the third round, we only linearize  $s - u_1$  ( $0 < u_1 < s$ ) S-boxes by guessing  $s - u_1$  output bits. Note that the output of the S-box in the third round is linear in  $K$ . In this way, we obtain  $3(s - u_1)$  linear equations in  $K$  because  $s - u_1$  S-boxes are linearized. Moreover, as mentioned above, such a guess for these  $s - u_1$  S-boxes also allows us to deduce  $3(s - u_1)$  quadratic equations in  $K$ . For the remaining  $u_1$  S-boxes, we can derive  $14u_1$  quadratic equations in  $K$ . In summary, we now obtain  $3(s - u_1)$  linear equations in  $K$  and  $6s + 3(s - u_1) + 14u_1 = 9s + 11u_1$  quadratic equations in  $K$ .

To solve these equations, we first apply Gaussian elimination to the  $3(s - u_1)$  linear equations to obtain  $3s - (3s - 3u_1) = 3u_1$  free variables. Then, we write the  $9s + 11u_1$  quadratic equations in these  $3u_1$  free variables. To solve these  $9s + 11u_1$  equations efficiently with the linearization technique, we constrain that

$$9s + 11u_1 \geq 3u_1 + 3u_1(3u_1 - 1)/2$$

i.e. the number of equations is larger than the number of variables after renaming the quadratic terms as new variables. The time complexity can therefore be estimated as

$$T_1 = 2^{2s+s-u_1} \cdot (3(s - u_1))^2 \cdot 3s + 2^{2s+s-u_1} \cdot (9s + 11u_1)^2 \cdot (3u_1 + 3u_1(3u_1 - 1)/2) \quad (9)$$

bit operations, where the term  $(3(s - u_1))^2 \cdot 3s$  represents the cost to apply Gaussian elimination to the  $3(s - u_1)$  linear equations in  $3s$  variables and the term  $(9s + 11u_1)^2 \cdot (3u_1 + 3u_1(3u_1 - 1)/2)$  represents the cost to solve the  $9s + 11u_1$  quadratic equations with the linearization technique. In Table 3, we show the optimal values of  $(s, u_1, T_1)$  to attack 3-round LowMC with full S-box layers. The memory complexity of this attack is very small, i.e. we only need to store the equation system, which costs about  $(9s + 11u_1) \cdot (3u_1 + 3u_1(3u_1 - 1)/2)$  bits.

It can be found that the first attack is much slower than the generic fast exhaustive search attack using Gray code proposed in [BCC<sup>+</sup>10]. In the following, we will describe how to use a different guess strategy to make the attacks greatly outperform the fast exhaustive search attack. Moreover, it can be found later that the idea in the first attack will be used for LowMC with partial nonlinear layers, which allows to devise attacks more efficient than the MITM attacks [BBVY21].

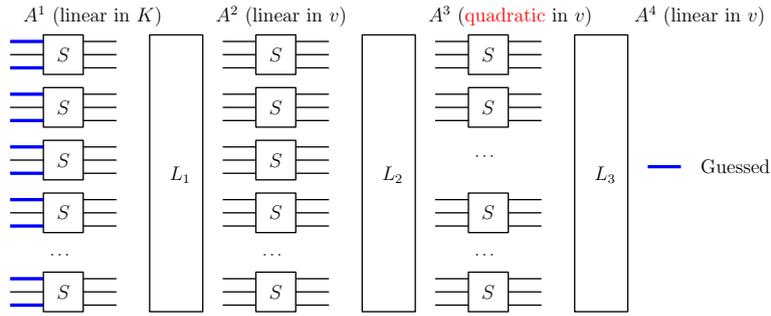
**Table 3:** Results for 3-round LowMC with full S-box layers

$n$	$k$	$s$	$r$	$u_1$	$T_1$	Memory
129	129	43	3	10	$2^{145.8}$	$2^{17.9}$
192	192	64	3	12	$2^{208.3}$	$2^{18.9}$
255	255	85	3	14	$2^{270.5}$	$2^{19.7}$

### 3.2 The Second Attack on LowMC

In the second attack, we use a rather naive guess strategy to linearize the 3-bit S-box. Specifically, we guess two input bits of the S-box rather than 1 output bit of the S-box to linearize it in the forward direction. However, different from Banik’s et al.’s attacks [BBDV20, BBVY21], we only apply this guess strategy for the first round.

For such a guess strategy, we directly obtain  $2s$  linear equations in  $K = (K_1, K_2, \dots, K_n)$ . After applying Gaussian elimination to these linear equations, we obtain  $s$  free variables  $v = (v_1, v_2, \dots, v_s)$  because  $n - 2s = 3s - 2s = s$  when LowMC uses the full S-box layers. Then, for the input state of S-box in the third round, it can be expressed as quadratic expressions in  $v$ . For the output state of the S-box in the third round, it is linear in  $K$  and thus can also be expressed as linear expressions in  $v$ . This can be seen from Fig. 2.



**Figure 2:** Illustration of the second attack on 3-round LowMC

At last, we consider the inverse of the S-box and obtain  $3s$  quadratic equations in these  $s$  variables  $v = (v_1, v_2, \dots, v_s)$ . We emphasize here that we should not consider the overdefined system of equations for the S-box here because 11 out of such 14 equations in  $v$  will be of degree higher than 2. The main reason is that the input bits are quadratic in  $v$  while the output bits are linear in  $v$ .

Therefore, the problem is reduced to solving  $3s$  quadratic equations in  $s$  variables. With the simple linearization technique, we can solve this system by guessing  $s - u_1$  variables and solve the  $3s$  quadratic equations in  $s - (s - u_1) = u_1$  variables with the linearization technique. This will require

$$3s \geq u_1 + u_1(u_1 - 1)/2.$$

The time complexity of the attack is then estimated as

$$T_2 = 2^{2s} \cdot 2^{s-u_1} \cdot (3s)^2 \cdot (u_1 + u_1(u_1 - 1)/2) = 2^{3s-u_1} \cdot (3s)^2 \cdot (u_1 + u_1(u_1 - 1)/2) \quad (10)$$

bit operations, i.e. updating the  $3s$  quadratic equations for each guess costs less time than solving  $3s$  quadratic equations in  $u_1$  variables.

In Table 4, we give the optimal values of  $(s, u_1, T_2)$  to attack 3-round LowMC. The memory complexity of this attack can be simply estimated as  $3s \cdot (s + s(s - 1)/2)$  bits, which is the cost to store the  $3s$  quadratic equations in  $s$  variables.

**Table 4:** Results for 3-round LowMC with full S-box layers

$n$	$k$	$s$	$r$	$u_1$	$T_2$	Memory
129	129	43	3	15	$2^{134.9}$	$2^{16.9}$
192	192	64	3	19	$2^{195.7}$	$2^{18.6}$
255	255	85	3	22	$2^{257.0}$	$2^{19.8}$

**Improving the time complexity with the crossbred-like algorithm [BDT22].** In the above attack, the problem is reduced to solving  $3s$  quadratic equations in  $s$  variables. We now show that we can use the crossbred-like algorithm for this system of quadratic equations to slightly improve the time complexity. First, we split the  $s$  variables  $v = (v_1, v_2, \dots, v_s)$  into  $y = (y_1, y_1, \dots, y_{s-u_1}) = (v_{u_1+1}, v_{u_1+2}, \dots, v_s)$  and  $z = (z_1, z_2, \dots, z_{u_1}) = (v_1, v_2, \dots, v_{u_1})$ . Then, as discussed in [Subsection 2.3](#), we can rewrite the  $3s$  quadratic equations in the following form:

$$A \cdot (z_1 z_2, z_1 z_3, \dots, z_{u_1-1} z_{u_1}, z_1, z_2, \dots, z_{u_1})^T = B.$$

We choose  $u_1$  such that

$$3s - u_1(u_1 - 1)/2 = u_1 + \epsilon, \epsilon > 0.$$

To optimize the time complexity, we choose  $u_1$  such that  $\epsilon$  is minimized. According to the complexity analysis described in [Subsection 2.3](#), the time complexity to solve the  $3s$  quadratic equation in  $s$  variables with this method is estimated as about

$$(3s)^2 \cdot \binom{s}{\leq 2} + 2^{s-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + s)$$

bit operations.

The time complexity of our attack on 3-round LowMC is thus estimated as

$$T_3 = 2^{2s} \cdot (3s)^2 \cdot \binom{s}{\leq 2} + 2^{3s-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + s)$$

bit operations. The optimal values of  $(s, u_1, \epsilon, T_3)$  are given in [Table 5](#). The memory complexity of this attack can be simply estimated as  $3s \cdot \binom{s}{\leq 2}$  bits, which is the cost to store the  $3s$  quadratic equations in  $s$  variables.

**Table 5:** Results for 3-round LowMC with full S-box layers

$n$	$k$	$s$	$r$	$(u_1, \epsilon)$	$T_3$	Memory
129	129	43	3	(15, 9)	$2^{127.2}$	$2^{16.9}$
192	192	64	3	(19, 2)	$2^{186.2}$	$2^{18.6}$
255	255	85	3	(22, 2)	$2^{246.8}$	$2^{19.8}$

### 3.3 The Third Attack on LowMC

In this part, we discuss the attacks on 4-round LowMC by combining the GnD technique and Dinur's algorithm [Din21] to solve nonlinear Boolean equations. As already shown above, if we linearize the round function of LowMC with Banik et al.'s guess strategy, we can only obtain quadratic equations in the key variables from the guess and these equations cannot be used to reduce the number of variables in the equation system. Moreover, linearizing 1 round of LowMC in this way requires to guess  $s$  bits and the problem is still

to solve nonlinear equations in  $3s$  variables. However, we can easily check that the time complexity to solve nonlinear equations in  $3s$  ( $s \in \{43, 64, 85\}$ ) variables with Dinur's algorithm is much larger than  $2^{2s}$  and thus the total time complexity will be much larger than  $2^{3s}$ . In other words, Banik et al.'s guess strategy is not suitable when we want to combine the GnD technique with Dinur's algorithm to improve the attacks.

Therefore, we use the naive guess strategy to linearize 1-round LowMC by guessing two input bits of each S-box. Specifically, for the first round, we guess  $2s$  input bits to linearize it. Then, apply Gaussian elimination on the  $2s$  linear equations in  $K = (K_1, K_2, \dots, K_n)$  to obtain  $n - 2s = 3s - 2s = s$  free variables  $v = (v_1, v_2, \dots, v_s)$ . In this way, we can write the input state of the S-box in the third round as quadratic expressions in  $v$  and write the output state of the S-box in the third round as quadratic expression in  $v$  as well.

Consider the  $i$ -th S-box ( $0 \leq i < s$ ) in the third round and we denote the polynomials in  $v$  to represent its input and output by  $p_{3i+1}(v), p_{3i+2}(v), p_{3i+3}(v), p_{3i+4}(v), p_{3i+5}(v), p_{3i+6}(v)$ , respectively. Note that all these polynomials are of degree 2. According to the definition of the S-box, we can construct the following 3 degree-4 polynomials in  $v$  denoted by  $q_{3i+1}(v), q_{3i+2}(v), q_{3i+3}(v)$ , respectively:

$$\begin{aligned} q_{3i+1}(v) &= p_{3i+4}(v) \oplus p_{3i+1}(v) \oplus p_{3i+2}(v)p_{3i+3}(v), \\ q_{3i+2}(v) &= p_{3i+5}(v) \oplus p_{3i+1}(v) \oplus p_{3i+2}(v) \oplus p_{3i+1}(v)p_{3i+3}(v), \\ q_{3i+3}(v) &= p_{3i+6}(v) \oplus p_{3i+1}(v) \oplus p_{3i+2}(v) \oplus p_{3i+3}(v) \oplus p_{3i+1}(v)p_{3i+2}(v). \end{aligned}$$

A naive upper bound for  $\text{Deg}(q_{3i+1} \oplus 1)(q_{3i+2} \oplus 1)(q_{3i+3} \oplus 1)$  is  $3 \cdot 4 = 12$ . However, a detailed look suggests that  $\text{Deg}(q_{3i+1} \oplus 1)(q_{3i+2} \oplus 1)(q_{3i+3} \oplus 1) \leq 2 \cdot 4 = 8$ , which has been exploited in Dinur's attacks on an odd number of rounds of LowMC [Din21].

In total, we can construct  $3s$  such degree-4 polynomials in  $v$ . The problem is reduced to solving these  $3s$  degree-4 equations in  $s$  variables  $v = (v_1, v_2, \dots, v_s)$ . Due to the high degree, our strategies in the first 2 attacks are not useful because the cost is very high. Hence, we consider Dinur's algorithm for this system of equations.

Similar to Dinur's attacks [Din21] on an odd number of rounds of LowMC, we consider the equations imposed by  $\lceil \frac{\ell}{3} \rceil$  S-boxes in the third round. For convenience, let

$$\ell = 3\ell_0 + \varepsilon, 0 \leq \varepsilon < 3.$$

In this way, we consider the  $3\ell_0$  equations  $\{q_i(v) = 0, 1 \leq i \leq 3\ell_0\}$  imposed by  $\ell_0$  S-boxes among the  $\lceil \frac{\ell}{3} \rceil$  S-boxes and  $\varepsilon$  equations  $\{q_i(v) = 0, 3\ell_0 + 1 \leq i \leq \ell\}$  imposed by the remaining  $\lceil \frac{\ell}{3} \rceil - \ell_0$  S-box, where  $\text{Deg}(q_i) = 4$  for  $1 \leq i \leq \ell$ .

Let

$$\tilde{F}(v) = (q_1(v) \oplus 1)(q_2(v) \oplus 1) \cdots (q_\ell(v) + 1).$$

Due to the structure of equations discussed above, we have that

$$d_{\tilde{F}} = \text{Deg}(\tilde{F}) = 8\ell_0 + 4\varepsilon.$$

To utilize Dinur's algorithm [Din21], we need to split the  $s$  variables  $v = (v_1, v_2, \dots, v_s)$  into  $y = (y_1, y_1, \dots, y_{s-u_1}) = (v_{u_1+1}, v_{u_1+2}, \dots, v_s)$  and  $z = (z_1, z_2, \dots, z_{u_1}) = (v_1, v_2, \dots, v_{u_1})$ , where  $u_1 = \ell - 1$ .

The assumption to correctly recover  $v$  becomes that for the correct guess<sup>4</sup> of  $y$ , there is at most 1 solution of  $z$  satisfying  $\{P_i(v) = P_i(y, z) = 0, 1 \leq i \leq \ell\}$ . Note that for each specified  $y$ , we are considering  $\ell = u_1 + 1$  equations in  $u_1$  variables. Hence, this assumption holds with a high probability. Later, we will use experiments to verify this assumption.

<sup>4</sup>For the wrong guess, we indeed do not care about the correctness of the assumption because the solution is always invalid. We only need to ensure when  $y$  is exactly the correct value, the assumption holds so that we can correctly compute  $z$ .

Moreover, similar to Dinur’s attacks [Din21], we prepare 4 different sets of  $\ell$  polynomials, each set of polynomials is constructed in the above way to exploit the structure of the equations caused by the S-box. This is used to amortize the cost to check the solution  $(y, z)$  against the  $3s$  equations. Specifically, for each guessed  $y = \hat{y}$ , we compute the corresponding  $z = \hat{z}$  if there is for each of the 4 sets. Then, when the same  $(\hat{y}, \hat{z})$  appears more than twice, we treat  $(\hat{y}, \hat{z})$  as a potential solution and check its correctness against the  $3s$  equations<sup>5</sup>. Otherwise, we simply abandon all the suggested  $(\hat{y}, \hat{z})$ . Later, we will use experiments to simulate its success probability.

According to the complexity analysis described in Subsection 2.4, the time complexity of our attack is estimated as

$$T_4 = 2^{2s} \cdot 4 \cdot (2 \cdot 4 \cdot \log_2 s \cdot 2^{u_1} \cdot \binom{s - u_1}{\leq d_{\hat{F}} - u_1 + 1}) + (u_1 + 1) \cdot (s - u_1) \cdot 2^{s - u_1}$$

bit operations. The memory complexity is estimated as

$$M_0 = 4 \cdot \left(\frac{1}{2}\right) \cdot \binom{s - u_1}{\leq d_{\hat{F}} - u_1 + 1} + (u_1 + 1) \cdot 2^{s - u_1}$$

bits if we use the standard Möbius transform. If using Dinur’s memory-efficient Möbius transform, the memory complexity is estimated as

$$M'_0 = 4 \cdot \left(\frac{1}{2}\right) \cdot \binom{s - u_1}{\leq d_{\hat{F}} - u_1 + 1} + (u_1 + 1) \cdot \binom{s - u_1}{\leq d_{\hat{F}} - u_1 + 1}.$$

The values of  $(s, u_1, \ell, d_{\hat{F}}, T_4, M_0, M'_0)$  are given in Table 6 to optimize the attacks on 4-round LowMC.

**Table 6:** Results for 4-round LowMC with full S-box layers

$n$	$k$	$s$	$r$	$(u_1, \ell, d_{\hat{F}})$	$T_4$	$M_0$	$M'_0$
129	129	43	4	(5, 6, 16)	$2^{133.8}$	$2^{42.6}$	$2^{36.7}$
192	192	64	4	(8, 9, 24)	$2^{195.0}$	$2^{61.2}$	$2^{53.4}$
255	255	85	4	(11, 12, 32)	$2^{255.8}$	$2^{79.6}$	$2^{68.0}$

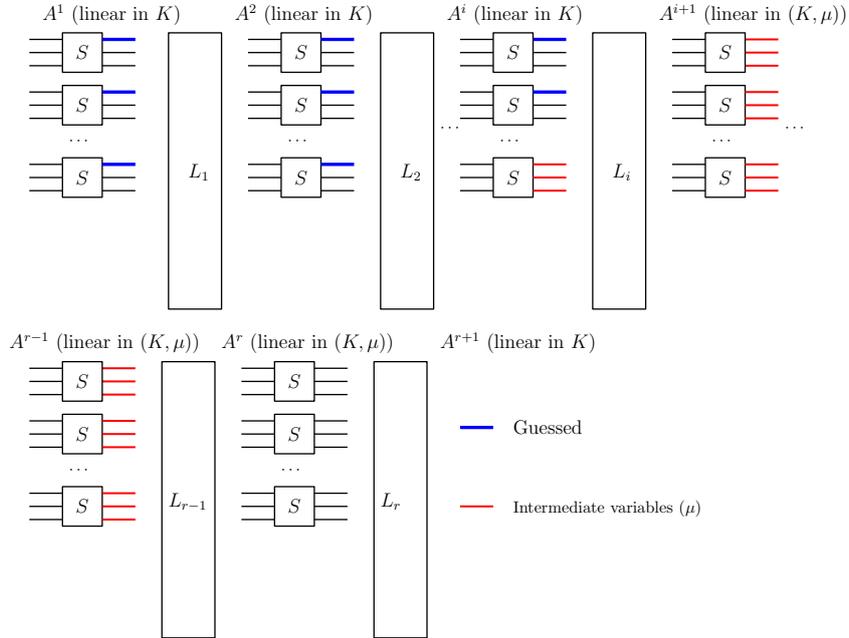
**Remark.** We note that there is a trivial time-memory tradeoff for Dinur’s algorithm by guessing variables. Specifically, to solve equations of degree  $d$  in  $u$  variables, Dinur’s algorithm generally requires  $u^2 \cdot 2^{(1-1/2.7d)u}$  bit operations and  $u^2 \cdot 2^{(1-1/1.35d)u}$  bits of memory. For attacks on 4-round LowMC,  $d = 4$  and  $u = k$ . For the naive guess strategy, we guess  $u - u_1$  variables and use Dinur’s algorithm to solve degree-4 equations in  $u_1$  variables, whose time and memory complexity become  $2^{u - u_1} \cdot u_1^2 \cdot 2^{(1-1/2.7d)u_1}$  and  $u_1^2 \cdot 2^{(1-1/1.35d)u_1}$ , respectively. To obtain time complexity not higher than that of our attacks, we find that the required memory complexity is larger than  $2^{84.6}$ ,  $2^{108.2}$  and  $2^{134.2}$  for the parameters  $k = 129$ ,  $k = 192$  and  $k = 255$ , respectively. Hence, our attacks can achieve much better tradeoffs. The main reason is that by guessing  $2s$  key variables to linearize the first round, the generic complexity of Dinur’s algorithm can be optimized by taking some useful properties of the equations into account. However, by randomly choosing  $2s$  key variables for guess, no properties of the degree-4 equations can be exploited.

<sup>5</sup>We can choose 4 different sets of equation systems in the way that the used S-boxes for one equation system are not repeatedly used in other 3 equation systems. In this way, for a wrong guess of the  $2s$  key bits to linearize the first round, it hardly appears that more than 2 systems will suggest the same isolated solution.

## 4 Attacks on LowMC with Partial Nonlinear Layers

In this part, attacks on LowMC with parameters  $s = \{1, 10\}$ ,  $r = \lfloor \frac{n}{s} \rfloor$  and  $n = k \in \{128, 192, 256\}$  will be taken into account, which are the targets listed in the LowMC competition. The best attacks on these parameters are achieved with the MITM technique [BBVY21]. The main idea of our attacks is very simple, i.e. we exploit the overdefined system of equations for the 3-bit S-box as we do in the first attack on LowMC.

For  $r$  rounds of LowMC, there are in total  $s(r-1)$  S-boxes in the first  $r-1$  rounds. First, we linearize the first  $\lambda$  S-boxes by guessing 1 output bit of each of these S-boxes in the forward direction. For the remaining  $s(r-1) - \lambda$  S-boxes in the first  $r-1$  rounds, for each of its three output bits, we introduce 3 intermediate variables to represent them. Hence, there are in total  $3s(r-1) - 3\lambda$  intermediate variables and they are denoted by  $\mu = (\mu_1, \mu_2, \dots, \mu_{3s(r-1)-3\lambda})$ . In this way, each input state of each round is linear in  $(\mu, K)$ , as shown in Fig. 3.



**Figure 3:** Illustration of the attacks on LowMC with partial nonlinear layers

Hence, for the input and output states of the last nonlinear layer, they can be written as linear expressions in  $n + 3s(r-1) - 3\lambda$  variables

$$(\mu, K) = (\mu_1, \mu_2, \dots, \mu_{3s(r-1)-3\lambda}, K_1, K_2, \dots, K_n),$$

respectively. By first ignoring the  $s$  S-boxes in the last nonlinear layer, we can construct  $n - 3s$  linear equations in these  $n + 3s(r-1) - 3\lambda$  variables  $(\mu, K)$ . Applying Gaussian elimination to this linear equation system will allow us to obtain

$$n + 3s(r-1) - 3\lambda - (n - 3s) = 3(sr - \lambda)$$

free variables. Denote these free variables by  $v = (v_1, v_2, \dots, v_{3(sr-\lambda)})$ . The time complexity of this Gaussian elimination is estimated as

$$T_5 = (n - 3s)^2 \cdot (n + 3s(r-1) - 3\lambda)$$

bit operations.

Now, we count the number of quadratic equations in  $v$ . As already mentioned at the first attack on **LowMC**, if we linearize 1 S-box by guessing an output bit, we can obtain 3 quadratic equations in its input. Hence, for our guess of  $\lambda$  bits,  $3\lambda$  quadratic equations in  $v$  can be constructed. Moreover, for the remaining  $s(r-1) - \lambda + s = sr - \lambda$  S-boxes in  $r$  rounds of **LowMC**, we can construct  $14(sr - \lambda)$  quadratic equations in  $v$ . In other words the problem now is reduced to solving  $14(sr - \lambda) + 3\lambda = 14sr - 11\lambda$  quadratic equations in  $3(sr - \lambda)$  variables.

**Using the crossbred-like algorithm.** We use the crossbred-like algorithm for this problem. Specifically, we first split  $v$  into  $z = (z_1, z_2, \dots, z_{u_1}) = (v_1, v_2, \dots, v_{u_1})$  and  $y = (y_0, y_1, \dots, y_{3(sr-\lambda)-u_1}) = (v_{u_1+1}, v_{u_1+2}, \dots, v_{3(sr-\lambda)})$ . Then, we find the minimal positive integer  $\epsilon$  such that

$$14sr - 11\lambda - u_1(u_1 - 1)/2 = u_1 + \epsilon, \epsilon > 0$$

Then, we iterate all values of  $y$  with Gray code and compute the corresponding  $z$  with Gaussian elimination, which will take

$$T_6 = (14sr - 11\lambda)^2 \cdot \binom{3(sr - \lambda)}{\leq 2} + 2^{3(sr-\lambda)-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + 3(sr - \lambda))$$

bit operations. The total time complexity of our attacks on  $r$  rounds of **LowMC** is thus estimated as

$$T_7 = 2^\lambda \cdot (T_5 + T_6)$$

bit operations. The memory complexity is negligible, which is dominated by storing the quadratic equations. Hence, the memory complexity is estimated as

$$M_1 = (14sr - 11\lambda) \cdot \binom{3(sr - \lambda)}{\leq 2}$$

The values of  $(\lambda, u_1, \epsilon, T_7, M_1)$  to optimize the attacks on **LowMC** with different parameters are given in [Table 7](#).

**Table 7:** Results for  $r = \lfloor \frac{n}{s} \rfloor$  rounds of **LowMC** with  $s$  S-boxes per round

$n$	$k$	$s$	$r$	$(\lambda, u_1, \epsilon)$	$T_7$	$M_1$
128	128	1	128	(114, 32, 10)	$2^{142.3}$	$2^{18.9}$
192	192	1	192	(175, 38, 22)	$2^{205.8}$	$2^{19.9}$
256	256	1	256	(238, 43, 20)	$2^{268.7}$	$2^{20.5}$
128	128	10	12	(106, 31, 18)	$2^{134.6}$	$2^{18.8}$
192	192	10	19	(173, 38, 16)	$2^{203.7}$	$2^{20.0}$
256	256	10	25	(231, 43, 13)	$2^{262.8}$	$2^{20.6}$

## 5 Experiments

We conduct 3 different experiments to verify the correctness of our attacks.

The first experiment is to verify our best attacks on 3-round **LowMC**. We choose the **LowMC** instance with the parameter  $(n, k, r) = (129, 129, 3)$  as the target. The main concern in our best 3-round attack is whether the crossbred-like algorithm can correctly work. Experiments have shown that it works as expected, and after guessing  $2 \times 43 + (43 - 15) = 96$

key bits, we can always reduce the problem to solving  $15 + 9 = 24$  linear equations in 15 variables  $z$ . Among 10000 random guesses, we find that only 21 solutions of  $z$  are suggested, thus resulting in a filtering probability of  $0.0021 \approx 2^{-8.9}$ , which is almost the same as the expected filtering probability  $2^{-9}$ . As already mentioned, checking the suggested solution is cheap by evaluating randomly picked quadratic equations not included in the 24 equations. For the correct guess, we find that the key can be correctly recovered.

The second experiment is to verify the assumptions used in our attacks on 4-round LowMC. Specifically, for the chosen 4 different sets of equation systems, when the key bits are correctly guessed, how many sets will suggest the correct isolated solution (the correct full key)? Moreover, when the key bits are wrongly guessed, how many sets will suggest the same isolated solution? We choose the LowMC instance with the parameter  $(n, s, r) = (129, 43, 4)$  as the target. For this target, we need to consider 4 different equation systems derived from 8 different S-boxes, where each equation system has 6 degree-4 equations in 5 variables and it corresponds to the equations deduced from 2 S-boxes. We performed 20000 random guesses by considering 4 equation systems simultaneously, i.e. there will be in total 80000 different equation systems. We find that there is at most 1 solution for 72000 equation systems, thus indicating that the assumption for the isolated solution holds with probability of about 0.9. In addition, among these 20000 random guesses, there are only 339 guesses such that more than 2 equation systems can suggest the same unique solution, thus resulting in a filtering probability of about  $339/20000 = 2^{-6.2}$ . We also perform 20000 experiments by always correctly guessing the key bits. It is found that there are about 16700 experiments such that more than 2 equation systems can suggest the same unique solution (the correct key). All in all, the attack succeeds with probability of about  $16700/20000 = 0.83$  and the filtering probability is low enough to amortize the time complexity to check the suggested solutions. Indeed, according to the experiments, if we check the solution only when more than 3 equation systems suggest the same isolated solution, the filtering probability becomes about  $1/20000$  and the success probability becomes about 0.5.

The last experiment is to verify the correctness of our attacks on LowMC with partial nonlinear layers. As the correctness of the crossbred-like algorithm has been verified in our 3-round attack, the main concern in this attack is whether we can obtain the expected number of linearly independent quadratic equations. We choose the LowMC instance with the parameter  $(n, s, r) = (128, 1, 128)$  as the target. In this attack, we expect that after guessing  $\lambda = 114$  bits, we can obtain  $14sr - 11\lambda = 14 \times 128 - 11 \times 114 = 538$  linearly independent quadratic equations in  $3(sr - \lambda) = 3 \times (128 - 114) = 42$  variables. Then, we solve these equations with the crossbred-like algorithm using the splitting parameter  $u_1 = 32$ , i.e. solving  $u_1 + \epsilon = 32 + 10 = 42$  linear equations in 32 variables for each guess of the  $42 - 32 = 10$  variables. Experiments show that the 538 quadratic equations are always linearly independent and we can always construct 42 linear equations in 32 variables for each guess of the 10 variables. For the correct guess, the key can be correctly recovered. Hence, the correctness of our attacks is verified.

## 6 Conclusion

While intuitively linearizing one LowMC S-box by guessing only one quadratic polynomial seems efficient, we show that naively guessing two linear polynomials to achieve the linearization can make the attacks on LowMC with full S-box layers much better. The main advantage of this naive guess strategy comes from the great reduction in the number of unknowns because guessing 1 linear polynomial directly reduces the number of unknowns by 1. Based on this new guess strategy, we can improve the key-recovery attacks on 3 and 4 rounds of LowMC using better time-memory tradeoffs than Dinur's algorithm.

Another contribution is to take advantage of the guessed quadratic polynomials and

the overdefined system of quadratic equations for the LowMC S-box to devise more efficient attacks on LowMC with partial nonlinear layers. In this way, recovering the full key is reduced to solving a much overdefined system of quadratic equations with the crossbred-like algorithm.

In conclusion, we have significantly improved the attacks on LowMC by using better time-memory tradeoffs and we expect this work further advances the understanding of the security of LowMC in the Picnic setting.

**Acknowledgement.** We thank the reviewers of ToSC 2022 Issue 3 for providing many useful comments to improve the quality of this paper. Especially, we thank André Schrottenloher for shepherding this paper. We also thank Itai Dinur for providing some useful advice on the preliminary version of this paper. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031 and Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141). This research was in part conducted under a contract of “Research and development on new generation cryptography for secure wireless communication service” among “Research and Development for Expansion of Radio Wave Resources (JPJ000254)”, which was supported by the Ministry of Internal Affairs and Communications, Japan.

## References

- [ARS<sup>+</sup>15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.
- [BBDV20] Subhadeep Banik, Khashayar Barooti, F. Betül Durak, and Serge Vaudenay. Cryptanalysis of LowMC instances using single plaintext/ciphertext pair. *IACR Trans. Symmetric Cryptol.*, 2020(4):130–146, 2020.
- [BBVY21] Subhadeep Banik, Khashayar Barooti, Serge Vaudenay, and Hailun Yan. New Attacks on LowMC Instances with a Single Plaintext/Ciphertext Pair. In *ASIACRYPT (1)*, volume 13090 of *Lecture Notes in Computer Science*, pages 303–331. Springer, 2021.
- [BCC<sup>+</sup>10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast Exhaustive Search for Polynomial Systems in  $\mathbb{F}_2$ . In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.
- [BDT22] Charles Bouillaguet, Claire Delaplace, and Monika Trimoska. A Simple Deterministic Algorithm for Systems of Quadratic Polynomials over  $\mathbb{F}_2$ . In *SOSA*, pages 285–296. SIAM, 2022.
- [CDG<sup>+</sup>17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In *CCS*, pages 1825–1842. ACM, 2017.
- [CP02] Nicolas T. Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.
- [DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-Order Cryptanalysis of LowMC. In *ICISC*, volume 9558 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2015.

- [Din21] Itai Dinur. Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over  $\text{GF}(2)$ . In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021.
- [DLMW15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 535–560. Springer, 2015.
- [DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.
- [DS11] Itai Dinur and Adi Shamir. An Improved Algebraic Attack on Hamsi-256. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 88–106. Springer, 2011.
- [JV17] Antoine Joux and Vanessa Vitse. A Crossbred Algorithm for Solving Boolean Polynomial Systems. In *NuTMiC*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.
- [KZ20] Daniel Kales and Greg Zaverucha. Improving the Performance of the Picnic Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):154–188, 2020.
- [LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques. In *CRYPTO (3)*, volume 12827 of *Lecture Notes in Computer Science*, pages 368–401. Springer, 2021.
- [LPT<sup>+</sup>17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating Brute Force for Systems of Polynomial Equations over Finite Fields. In *SODA*, pages 2190–2202. SIAM, 2017.
- [LWM<sup>+</sup>22] Fukang Liu, Gaoli Wang, Willi Meier, Santanu Sarkar, and Takanori Isobe. Algebraic Meet-in-the-Middle Attack on LowMC. *IACR Cryptol. ePrint Arch.*, page 19, 2022.
- [RST18] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of Low-Data Instances of Full LowMCv2. *IACR Trans. Symmetric Cryptol.*, 2018(3):163–181, 2018.