IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 1, 29 pages.
https://doi.org/10.62056/ayzojbkrz

New Attacks on LowMC Using Partial Sets in the Single-Data Setting

Subhadeep Banik[1], Andrea Caforio[2] and Serge Vaudenay[2]

[1] Universita della Svizzera Italiana, Lugano, Switzerland
[2] Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland

Abstract. The LowMC family of block ciphers was proposed by Albrecht et al. in [ARS+15], specifically targeting adoption in FHE and MPC applications due to its low multiplicative complexity. The construction operates a 3-bit quadratic S-box as the sole non-linear transformation in the algorithm. In contrast, both the linear layer and round key generation are achieved through multiplications of full rank matrices over GF(2). The cipher is instantiable using a diverse set of default configurations, some of which have partial non-linear layers i.e., in which the S-boxes are not applied over the entire internal state of the cipher.
The significance of cryptanalysing LowMC was elevated by its inclusion into the NIST PQC digital signature scheme PICNIC in which a successful key recovery using a single plaintext/ciphertext pair is akin to retrieving the secret signing key. The current state-of-the-art attack in this setting is due to Dinur [Din21a], in which a novel way of enumerating roots of a Boolean system of equation is morphed into a key-recovery procedure that undercuts an ordinary exhaustive search in terms of time complexity for the variants of the cipher up to five rounds.
In this work, we demonstrate that this technique can efficiently be enriched with a specific linearization strategy that reduces the algebraic degree of the non-linear layer as put forward by Banik et al. [BBDV20]. This amalgamation yields new attacks on certain instances of LowMC up to seven rounds.

Keywords: LowMC · Block Cipher · Cryptanalysis

1 Introduction

The block cipher LowMC has been quite popular in the cryptographic community ever since it was proposed in 2015. The straightforward nature of the construction and its various configurations lend themselves well to a broad range of attack techniques. The announcement of the LowMC cryptanalysis challenge in 2020 (`https://lowmcchallenge.github.io/`) further incentivized the efforts across the community with the competition having been renewed twice since then.

The structure of LowMC is unique due to its simple component functions, solely consisting of a non-linear 3-bit S-box and a matrix multiplication over GF(2) that represents the linear layer. Note that, unlike some other block ciphers which apply S-boxes over the entire internal state, some instances of LowMC allow a partial application of the non-linear layer further lowering its multiplicative complexity. Henceforth, we will refer to these two design philosophies as LowMC instances with complete and partial non-linear layers.

Although the large canon of cryptanalytic results on LowMC is partly due to its broad range of possible instantiations, it was its integration into the NIST PQC digital signature scheme PICNIC that made the single plaintext/ciphertext pair setting an important attack

E-mail: subhadeep.banik@usi.ch (Subhadeep Banik), andrea.caforio@epfl.ch (Andrea Caforio), serge.vaudenay@epfl.ch (Serge Vaudenay)

Received: 2024-01-08 Accepted: 2024-03-05

target. This is because a successful key recovery attack on LowMC using only a single plaintext and ciphertext is equivalent to retrieving the signing key of PICNIC. Below, we give a brief overview of existing attacks in the single data-point setting. For a survey of other key recovery techniques, we refer the reader to [GKRS].

1.1 Previous Work

The first successful key recovery that only relies on a single plaintext/ciphertext pair was proposed by Banik et al. [BBDV20]. The authors used the fact that after guessing the value of any balanced quadratic Boolean function on the inputs of the LowMC S-box the transformation becomes completely linear. The authors chose the 3-variable majority function for this purpose, but they show that any balanced quadratic function can be used. Using this, they demonstrated various attacks on 2-round LowMC with complete non-linear layer, and $0.8 \cdot \lfloor \frac{n}{s} \rfloor$-round LowMC with partial non-linear layers. Here, n denotes the blocksize of the LowMC instance, and s denotes the number of S-boxes in each round (with $3s \leq n$).

The linearization method was used in [BBVY21] to extend the attack to 3-round variants with complete non-linear layer and $1 \cdot \lfloor \frac{n}{s} \rfloor$-round variants with partial non-linear layers, using two applications of a meet-in-the-middle procedure. Further in [LIM21], the authors proposed an algebraic attack on 3-round LowMC. The current state-of-the-art method was proposed by Dinur [Din21a] in which a newly devised scheme of finding roots to Boolean systems of equations is transformed into a cryptanalytic attack. In fact, given a plaintext/ciphertext pair, the paper transforms the problem of recovering the secret key into the problem of finding the common root of a set of n equations in n variables over GF(2). This attack is particularly well-suited for low-degree systems and thus was successfully applied to $\{2, 3, 4, 5\}$-round versions of LowMC where the S-box is performed over the entire internal state. However, the method is not suitable for LowMC instances with partial non-linear layers, since the number of rounds in such instances is generally considerably higher, and the degree of the internal state variables (as a function of the key) doubles every round. Recently, [LMSI22] used a variant of this technique to attack upto 4 rounds of LowMC in the single data setting, but their technique is not possible to extend to any higher number of rounds, as it would be well above the complexity of exhaustive search.

1.2 Contributions

In this paper, we combine the linearization techniques of [BBDV20, BBVY21] and the equation solving methods of [Din21a] to cryptanalyze LowMC instances with complete non-linear layers. The principal technique in the attack that we propose is to use linearization to transform the problem of finding the secret key into an equivalent problem of finding the common roots of an equation system with only around $2n/3$ variables. Thus, although our method requires time complexity of the same order as reported by [Din21a], we do it with significantly less memory than reported in [Din21a]. Our method, for all instances of LowMC, requires around $2^{2n/3}$ bits of memory, which is roughly the space required to solve an equation system over GF(2) in $2n/3$ variables. We also report the first attacks on LowMC instantiations upto 7 rounds in single data setting.

The main idea is as follows: we know that the LowMC S-box can be completely linearized by guessing the value of any balanced quadratic function in its input bits. Since the master key (xored with the known plaintext) is directly input to the S-box layer of the first LowMC round, by guessing the values of some balanced quadratic equation in the key bits we can directly linearize the first round, which serves to reduce the algebraic degree of the polynomial equations relating the plaintext and ciphertext. What this also does is partition the key space (which is $\{0, 1\}^n$ for complete LowMC instances with blocksize equal to n bits) into disjoint sets, depending on the value of the guessed function. For example, if the key space is of size 12 bits and we use the 3-variable majority function (**maj**) for linearization, i.e. by guessing the $12/3 = 4$ values of $g_i = \mathbf{maj}(k_{3i}, k_{3i+1}, k_{3i+2})$ (for $i = 0, 1, 2, 3$), then note that we have partitioned the key space into 2^4 disjoint sets, each of which is indexed by the guess vector $[g_3, g_2, g_1, g_0] \in \{0, 1\}^4$ and has size 2^8. Generalizing this, we can say that this process of linearizing, partitions the key space into $2^{n/3}$ disjoint sets each of size $2^{2n/3}$ (which we call "partial sets/space" interchangeably throughout the paper).

Much of the technical content in the paper deals with how to perform efficient arithmetic over these partial sets. It is well known that to evaluate the truth table of a Boolean function in n variables and algebraic degree $d \leq n$, then we need the evaluation of the function on $\sum_{i=0}^{d} \binom{n}{i}$ points of its input space. One of the main results in this paper, is to show that if we needed to evaluate the function on any one of these partial sets then we need the function evaluation on a much smaller set of points. Most of the optimizations that we have derived in the paper in terms of time and space complexity, stems from this key observation. As a result, we were able to attack some 5, 6 and 7 round instances of LowMC, with essentially memory less than or around $2^{2n/3}$ bits. A complete list of results is presented in Table 1. We also compare our results with that of [BCC+10], which outlines a method of finding a common root of an equation system in n unknowns and degree d over $GF(2)$ using a Gray-code based traversal of the solution space, and requires polynomial memory to execute. This method takes around $2d \cdot \log_2 n \cdot 2^n$ bit-operations and n^d bits of memory. Our method does not always outperform the Gray-code assisted exhaustive search complexity (e.g. $n = 129, R = 6, 7$). We only report those instances when our complexity is better than the gray-code assisted exhaustive search with a constant probability of success (at least 0.5).

1.3 Organization of the Paper

In Section 2, we present some preliminary introduction to the algebraic structure of LowMC, and the LowMC cryptanalysis challenge. Section 3 presents some initial ideas about linearization, and how it helps set up the attack on the various LowMC instances with both even and odd number of rounds. Section 4 relates to the problem of efficiently evaluating a Boolean function over any one of the partial sets defined above. The remaining part of Section 5 presents the mathematical details of the attack, and explicit derivations of the time and space complexity. In Section 6 we study generic time-memory trade-offs that can further decrease the memory complexity. We also compare how these type of trade-offs affect our attack in comparison with [Din21a]. We conclude the paper in Section 7.

2 Preliminaries

The LowMC round function is a typical SPN construction given in Fig. 1. It consists of an n-bit block undergoing either a partial or a complete substitution layer consisting of s 3-bit S-boxes where $3s \leq n$. It is followed by an affine layer which consists of multiplication of the block with an invertible $n \times n$ matrix over $\mathbb{F}_2$ and addition with an n-bit round constant. Finally, the block is xored with an n-bit round key. If the master secret key K is of size n-bits (which is true for all the instances in the LowMC challenge), then each round key is obtained by multiplication of K with an $n \times n$ invertible matrix over GF(2). As in most SPN constructions, the plaintext is first xored with a whitening key which for LowMC is simply the secret key K, and the round functions are executed R times to give the ciphertext. From the point of view of cryptanalysis, we note that the design is completely known to the attacker, i.e., all the matrices and constants used in the round function and key update are known. Note that in general instantiations of LowMC, the key size and block size are not the same. The whitening key and all the round keys are extracted by multiplying the master key with full rank matrices over GF(2). However

Table 1: Summary of results. R denotes the number of rounds. Note the time complexity (TC) and memory complexity (MC) are given in number of bit operations and bits respectively. The complexity for exhaustive search is computed as per Lemma 1. We further compare our results with the gray code assisted exhaustive search technique proposed in [BCC+10], using the expression in Lemma 2. (n_1, ℓ, N, θ) are parameters used in the attack which are explained in Section 5. Success probability has been calculated as per Equation (10).

R	n	s	(n_1, ℓ, N, θ)	Success Prob.	TC	MC	Exhaustive Search	Gray code [BCC+10]	Reference
2	129	43		1.00	2^{107}	$O(1)^{**}$	2^{145}	2^{134}	[BBDV20]*
	129	43		1.00	2^{97}	2^{53}			[BBVY21]*
	129	43		$0.68^{\dagger}$	2^{118}	2^{92}			[Din21a]
	129	43	$(18, 15, 14, 10)$	0.91	$2^{125.9}$	$2^{78.3}$			This paper
	192	64		1.00	2^{151}	$O(1)$	2^{209}	2^{197}	[BBDV20]
	192	64		1.00	2^{139}	2^{75}			[BBVY21]
	192	64		0.68	2^{170}	2^{126}			[Din21a]
	192	64	$(30, 23, 24, 15)$	1.00	$2^{182.8}$	$2^{113.1}$			This paper
	255	85		1.00	2^{194}	$O(1)$	2^{273}	2^{260}	[BBDV20]
	255	85		1.00	2^{182}	2^{97}			[BBVY21]
	255	85		0.68	2^{222}	2^{173}			[Din21a]
	255	85	$(39, 30, 16, 11)$	1.00	$2^{238.9}$	$2^{148.4}$			This paper
3	129	43		1.00	2^{140}	2^{53}	2^{146}	2^{135}	[BBVY21]
	129	43		0.68	2^{125}	2^{104}			[Din21a]
	129	43		0.83	$2^{127.2}$	$2^{16.9}$			[LMSI22]
	129	43	$(12, 10, 20, 14)$	0.62	$2^{130.5}$	$2^{82.7}$			This paper
	192	64		1.00	2^{204}	2^{75}	2^{210}	2^{198}	[BBVY21]
	192	64		0.68	2^{180}	2^{150}			[Din21a]
	192	64		0.83	$2^{186.2}$	$2^{18.6}$			[LMSI22]
	192	64	$(21, 16, 32, 23)$	0.59	$2^{188.9}$	$2^{119.3}$			This paper
	255	85		1.00	2^{267}	2^{97}	2^{274}	2^{261}	[BBVY21]
	255	85		0.68	2^{235}	2^{197}			[Din21a]
	255	85		0.83	$2^{246.8}$	$2^{19.8}$			[LMSI22]
	255	85	$(27, 21, 12, 10)$	0.55	$2^{246.9}$	$2^{156.1}$			This paper
4	129	43		0.68	2^{130}	2^{113}	2^{146}	2^{135}	[Din21a]
	129	43		0.83	$2^{133.8}$	$2^{36.7}$			[LMSI22]
	129	43	$(6, 7, 14, 8)$	1.00	$2^{133.9}$	$2^{86.2}$			This paper
	192	64		0.68	2^{188}	2^{164}	2^{210}	2^{198}	[Din21a]
	192	64		0.83	$2^{195.0}$	$2^{53.4}$			[LMSI22]
	192	64	$(12, 11, 10, 8)$	0.64	$2^{192.9}$	$2^{123.8}$			This paper
	255	85		0.68	2^{245}	2^{218}	2^{274}	2^{261}	[Din21a]
	255	85		0.83	$2^{255.8}$	$2^{68.0}$			[LMSI22]
	255	85	$(18, 15, 14, 10)$	0.91	$2^{252.9}$	$2^{162.2}$			This paper
5	129	43	$(3, 5, 12, 8)$	0.95	$2^{135.7}$	$2^{88.0}$	2^{146}	2^{136}	This paper
	192	64		0.68	2^{192}	2^{173}	2^{210}	2^{199}	[Din21a]
	192	64	$(9, 8, 32, 19)$	0.96	$2^{196.9}$	$2^{127.2}$			This paper
	255	85		0.68	$2^{254\ \dagger\dagger}$	2^{228}	2^{274}	2^{262}	[Din21a]
	255	85	$(12, 11, 12, 9)$	0.82	$2^{256.9}$	$2^{166.0}$			This paper
6	192	64	$(6, 6, 18, 13)$	0.52	$2^{198.0}$	$2^{128.5}$	2^{211}	2^{199}	This paper
	255	85	$(9, 8, 18, 13)$	0.52	$2^{259.7}$	$2^{168.51}$	2^{275}	2^{262}	This paper
7	255	85	$(6, 6, 17, 12)$	0.57	$2^{262.0}$	$2^{170.5}$	2^{275}	2^{263}	This paper

*The papers [BBDV20] and [BBVY21] report complexities in number of encryptions. We recalculate them in terms of number of bit-operations using the estimate in Lemma 1.

**$O(1)$ refers to constant memory required for only storing intermediate variables and running loops.

†The success probability in [Din21a] depends on a probability $\frac{1}{2}$ event occurring at least twice in 4 trials, the probability of which is around 0.68. †† Recomputed using additional $S_1 + S_2$ term in Sec 6.1.

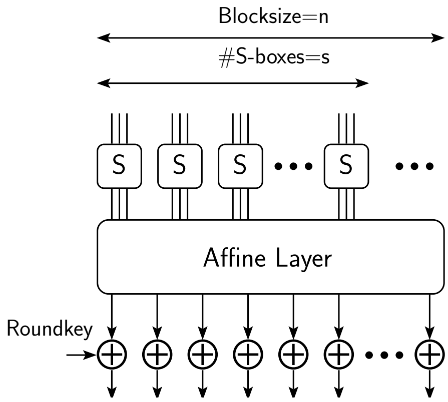


Figure 1: LowMC Round Function

for all the instances of LowMC used in the LowMC challenge the block size and key size are the same. This being so, the lengths of the master key, whitening key and all the subsequent round keys are the same. Effectively, this makes all these keys related to each other by multiplication with an invertible matrix over GF(2). Thus all round keys can be extracted by multiplying the whitening key with an invertible matrix. So for all practical purposes used in this paper, the whitening key can also be seen as the master secret key. This is true since given any candidate whitening key, all round keys can be generated from it, and thus given any known plaintext-ciphertext pair, it is possible to verify if that particular candidate key has been used to generate the corresponding plaintext/ciphertext pair. As such we use the terms master key/whitening key interchangeably.

The LowMC challenge specifies 9 challenge scenarios for key recovery given only 1 plaintext-ciphertext pair, i.e., for single-data complexity.

- **1**. $[n = 128,\ s = 1]$ **2**. $[n = 128,\ s = 10]$ **3**. $[n = 129,\ s = 43]$
- **4**. $[n = 192,\ s = 1]$ **5**. $[n = 192,\ s = 10]$ **6**. $[n = 192,\ s = 64]$
- **7**. $[n = 256,\ s = 1]$ **8**. $[n = 256,\ s = 10]$ **9**. $[n = 255,\ s = 85]$

The number of rounds R for instances with complete S-box layer is either 2, 3, or 4 and for instances with a partial S-box layer can vary between $0.8 \times \lfloor \frac{n}{s} \rfloor$, $\lfloor \frac{n}{s} \rfloor$ and $1.2 \times \lfloor \frac{n}{s} \rfloor$. When these are not integers, the number of rounds is taken as the next higher integer. The key length k for all instances is n bits. PICNIC v3.0 [Zav] incidentally uses LowMC instances with the parameter sets $[n, s, R]$ given by $[128, 10, 20]$, $[192, 10, 30]$, $[256, 10, 38]$ (partial S-box layer) and $[129, 43, 4]$, $[192, 64, 4]$, $[255, 85, 4]$ (complete S-box layer) for use under different security levels. It should be noted that our attack only targets the LowMC instances with complete non-linear layers, since all instances of partial non-linear layer based constructions have high algebraic degree due to the large number of rounds it executes.

3 Attack Preliminaries

Before we begin let us establish the number of bit operations required to perform an exhaustive search on an R-round instance of LowMC with complete non-linear layers given a single plaintext/ciphertext pair. Note that since the paper focuses solely on LowMC instances with complete S-box layers, for conciseness we will no longer use this term, and henceforth any mention of a LowMC instance should be understood as being with complete non-linear layers. The following lemma was proven in [BBVY21], but we restate it for completeness.

Lemma 1. *[BBVY21] Performing one encryption with any R round instance of LowMC requires around $2Rn^2$ bit-operations. And thus the cost of exhaustive search is around $Rn^2 \cdot 2^{n+1}$ bit-operations.*

Proof. Although this was shown in [BBVY21], we give a brief proof sketch for completeness. Any one round of LowMC requires 2 matrix-vector multiplications (between an $n \times n$ matrix and an $n \times 1$ vector) over GF(2): one to perform the linear layer on the state, and the second to generate the round key from the master key. This needs n^2 bit-operations each. One round key addition takes n bit-operations. One 3-bit s-box requires around 8 operations (see below) and so the entire substitution layer needs around $8n/3$ operations.

```
(1) P0=P1*P2, (2) S0=X0+P0, (3) T1=X0+X1, (4) P1=X0*X2.
(5) S1=X1+P1, (6) T2=T1+X2, (7) P2=X0*X1, (8) S2=T2+P2
```

Thus the total number of bit-operations for R-round LowMC is around $R(2n^2+n+8n/3) \approx 2Rn^2$ bit-operations, and so exhaustive search which requires 2^n encryptions needs $Rn^2 \cdot 2^{n+1}$ bit-operations

□

Lemma 2. *Accelerating exhaustive search for any R round instance of LowMC using Gray code based search of [BCC+10] takes around $2D \cdot \log_2 n \cdot 2^n$ bit-operations, where $D = 2^{\lceil R/2 \rceil}$.*

Proof. It was shown in [BCC+10] that a system of n Boolean polynomials of degree d in n variables can be solved using $2d \cdot \log_2 n \cdot 2^n$ bit-operations. All that's left to show that the R round instance of LowMC can be written as n polynomials of degree $2^{\lceil R/2 \rceil}$. Given any plaintext-ciphertext pair, computing forward, the state bits after $\lceil R/2 \rceil$ rounds can be written as polynomials in the key variables of degree $2^{\lceil R/2 \rceil}$. Similarly computing backwards from the ciphertext, the state bits after the $\lceil R/2 \rceil$-th forward round can be written as polynomials of degree $2^{R - \lceil R/2 \rceil} = 2^{\lfloor R/2 \rfloor}$ [1]. Equating these yields n polynomials of the required degree. □

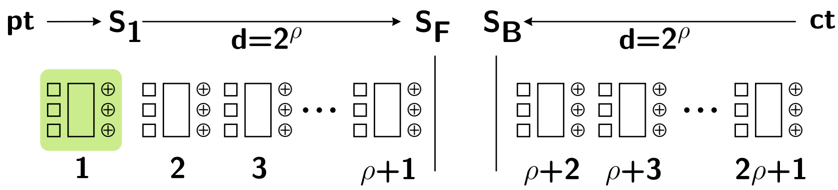
The starting point of the attack in [BBDV20] was the following lemma that helps linearize the LowMC S-box by guessing only one balanced quadratic expression on its input bits.

Lemma 3. *[BBDV20] Consider the LowMC S-box S defined over the input bits x_0, x_1, x_2. If we guess the value of any 3-variable quadratic Boolean function f which is balanced over the input bits of the S-box, then it is possible to re-write the S-box as affine function of its input bits.*

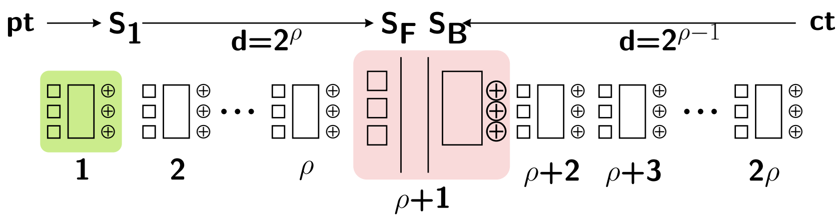
The authors used the majority function $f = x_0x_1 + x_1x_2 + x_0x_2$ for this purpose which is both quadratic and balanced. This is true since the LowMC S-box output bits can be written as:

$$\begin{aligned} s_0 &= x_0 + x_1 \cdot x_2 && = f \cdot (x_1 + x_2 + 1) + x_0, \\ s_1 &= x_0 + x_1 + x_0 \cdot x_2 && = f \cdot (x_0 + x_2 + 1) + x_0 + x_1, \\ s_2 &= x_0 + x_1 + x_2 + x_0 \cdot x_1 && = f \cdot (x_0 + x_1 + 1) + x_0 + x_1 + x_2. \end{aligned}$$

[1] This follows since both the forward and inverse S-box of LowMC are quadratic.



(a) R= 2ρ+1 is odd



(b) R= 2ρ is even

Figure 2: The attack setup for even and odd rounds for LowMC instances. The first round shown in green is linearized.

Using the above fact, the first attack proposed in [BBDV20] used only the linearization technique to obtain affine equations relating plaintext and ciphertext. The idea is as follows. The values of the majority function at the input of all the S-boxes in the encryption circuit were guessed: this made expression relating the plaintext and ciphertext completely linear in the key variables, i.e., of the form:

$$A \cdot [k_0, k_1, \ldots, k_{n-1}]^T = const, \tag{1}$$

where A is an $n \times n$ matrix over GF(2). Thereafter the key could be found by using Gaussian elimination. A wrong key found by this method could be discarded by recalculating the encryption and checking if the given plaintext mapped to the given ciphertext.

The above method would work if the total number of S-boxes in the encryption circuit is strictly less than the size of the key in bits. This happens for **a)** 2-round LowMC with complete non-linear layers and **b)** $0.8 \times \lfloor \frac{n}{s} \rfloor$-round LowMC with partial non-linear layers. For higher round instances of LowMC, this approach obviously takes complexity more than exhaustive search of the key and so becomes infeasible. In [BBVY21], the authors had shown how to combine meet-in-the-middle techniques along with linearization to extend the attack to 3-round LowMC with complete non-linear layers and $1 \times \lfloor \frac{n}{s} \rfloor$-round LowMC with partial non-linear layers. In this paper, we look to combine linearization with the equation solving techniques suggested by Dinur and show that we can attack up to 7 round instances of LowMC.

3.1 Attack Setup after Linearization

Let us outline the basic steps of the attack. Without loss of generality, consider the plaintext to be the all zero string, due to which the input to the first round s-box is the master key itself. We try to guess the values of some balanced quadratic expression in the keybits before it is input to the first round S-boxes. As a result of this the first round can be completely linearized, and the output bits S_1 of the first round are essentially affine expressions of the keybits.

3.1.1 Odd Number of Rounds:

For an arbitrary instance of LowMC with $R = 2\rho + 1$, i.e., odd number of rounds, we can form n equations of degree 2^ρ as follows:

1. Consider rounds 2 to $\rho + 1$ of LowMC. The function that maps the round 2 input to the $\rho + 1$-th round output is essentially a map of algebraic degree 2^ρ in the key, since there are ρ rounds in the map, each of degree 2.

2. Consider the inverse rounds $2\rho + 1$ down to $\rho + 2$. Again, there are ρ inverse rounds. The function that maps the round $2\rho + 1$ output (which is essentially the ciphertext) to the $\rho + 2$-th round input is also therefore a map of algebraic degree 2^ρ.

3. Since the algebraic degree of S_1 is 1, we can get n algebraic equations of degree 2^ρ by executing rounds 2 to $\rho + 1$ on S_1 to get S_F (see Figure 2). Each of the n bits of S_F is a Boolean polynomial in the key bits of degree 2^ρ. Similarly by executing the inverse rounds $2\rho + 1$ to $\rho + 2$ over the ciphertext, we get the state S_B. Each bit of S_B gives us another set of n equations of degree 2^ρ. Equating these 2 sets of expressions yields n equations of degree 2^ρ each.

3.1.2 Even number of rounds:

If the number of rounds $R = 2\rho$ is even, then we proceed as follows:

1. Consider rounds 2 to ρ of LowMC. The function that maps the round 2 input to the ρ-th round output is a map of algebraic degree $2^{\rho-1}$ in the key, for obvious reasons.

2. The function that maps the round 2ρ output (which is the ciphertext) to the $\rho+2$-th round input is also therefore a map of algebraic degree $2^{\rho-1}$.

3. We take S_F as the state after executing the substitution layer in round $\rho + 1$ (see Figure 2). Since S_1 is linear and a total of ρ substitution layers are executed to get S_F, we have that each bit of S_F is a Boolean polynomial in the keybits of degree 2^ρ.

4. We take S_B as the state just before the affine layer in round $\rho + 1$. Again, a total of $\rho - 1$ inverse substitution layers are executed to reach S_B from the ciphertext. Hence each bit in S_B is of degree $2^{\rho-1}$.

5. Equating each bit of S_F with the corresponding bit of S_B gives us n equations of degree 2^ρ each.

Although each equation is of degree 2^ρ, [Din21a] had pointed out an interesting property of these equations. Consider $p_i(k)$ to be the Boolean polynomial representing the i^{th} bit of S_B. Similarly let $a_i(k)$ represent the Boolean polynomial for the i^{th} bit of the state at the input of round $\rho + 1$, i.e., one substitution layer before S_F. Note that, due to the linearization step, each p_i, a_i have algebraic degree equal to $2^{\rho-1}$. It can be seen that the equation obtained after equating S_F and S_B, for any 3 consecutive bits aligned under the same S-box are as follows:

$$\begin{aligned} p_i(k) + a_i(k) + a_{i+1}(k)a_{i+2}(k) &= 0 \\ p_{i+1}(k) + a_i(k) + a_{i+1}(k) + a_i(k)a_{i+2}(k) &= 0 \\ p_{i+2}(k) + a_i(x) + a_{i+1}(k) + a_{i+2}(k) + a_i(k)a_{i+1}(k) &= 0 \end{aligned}$$

If we multiply the polynomials in the left side of three equations together, we get a polynomial whose degree is $4 \cdot 2^{\rho-1}$. This is much lower than $3 \cdot 2^\rho$ which is the expected degree of the product of 3 degree 2^ρ polynomials. We further observe that if only any 2 of these polynomials are multiplied together then the algebraic degree of the product is again only $3 \cdot 2^{\rho-1}$ which is again much lower than $2 \cdot 2^{\rho}$.

Since in the LowMC instances with complete non-linear layers, the keysize/blocksize is a multiple of 3, let $n = 3t$. Let $h : \{0,1\}^3 \to \{0,1\}$ be any 3-variable balanced quadratic Boolean function. Note that for all such h, we can linearize the first round if we guess $h(k_{3i}, k_{3i+1}, k_{3i+2})$ for all $i \in [0, t-1]$. We have already seen that after doing this we can derive n algebraic equations over GF(2) of degree 2^{ρ} each. Cryptanalysis of LowMC would essentially be equivalent to finding a common root of these equations.

4 Finding roots of an equation system over partial space

We can now solve the n equations obtained by linearizing the first round of LowMC, to find its roots and thus find the key, however with some caveat. Note that the equations were obtained by initially restricting the value of the master key to a specific subset of $\{0,1\}^{3t}$. For example let B^1 be the set of four 3-bit vectors over which h is 1, and B^0 be the complement of B^1. If the initial guess of h over the $3t$ bits of the key, is some vector $G = [g_{t-1}, g_{t-2}, \dots, g_0] \in \{0,1\}^t$ (i.e., $h(k_{3i}, k_{3i+1}, k_{3i+2}) = g_i$), then it only makes sense if the exhaustive search is done over the space $B^G = B^{g_{t-1}} \times B^{g_{t-2}} \times \cdots B^{g_0}$. The latter is a set of size $4^t = 2^{2n/3}$. Also note that there are exactly $2^t = 2^{n/3}$ (one for each value of G) of such sets and these sets partition $\{0,1\}^n$.

Consider the function h for which $B^0 = \{000,\ 010,\ 100,\ 111\}$ and $B^1 = \{001,\ 011,\ 101,\ 110\}$. It is clear that h is balanced, and it is easily verifiable that it is also quadratic. In fact it can be verified that $h = s_0$, the first output bit of the LowMC s-box. For convenience we write $B^0 = \{0, 2, 4, 7\}$ and $B^1 = \{1, 3, 5, 6\}$. With this background in hand, we will now discuss the finer details of the attack. Before we describe our technique, it would be instructive (at least for the completeness of the paper) to look at a few preliminary tools we will use to perform the attack.

Note that this section is devoted to the problem of efficiently evaluating a Boolean function over any one of the partial sets B^G defined above. It develops tools and also establishes bounds with respect to time and space complexity, that will be used when the attack is finally described in Section 5.

4.1 Evaluating a function over a partial space

So to begin, we guess the values of some balanced quadratic equation in all the key-triples before it is input to the first round S-boxes, and obtain n equations of degree $d = 2^{\rho}$ for any $R = 2\rho$ or $R = 2\rho + 1$-round instance of LowMC, as explained in the previous section. Since we can choose any balanced quadratic function for linearizing the S-box, let us choose the function h for which $B^0 = \{0, 2, 4, 7\}$ and $B^1 = \{1, 3, 5, 6\}$, as defined above. The reason we choose this function will be clear in a moment. Note that when B^0 is expressed as the following 4×3 matrix over GF(2),

$$B^0 : \begin{array}{ccc} u_i & & \mathbf{u}_i \\ 00 & \to & \\ 01 & \to & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ 10 & \to & \\ 11 & \to & \end{array} \qquad B^1 : \begin{array}{ccc} u_i & & \mathbf{u}_i \\ 00 & \to & \\ 01 & \to & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \\ 10 & \to & \\ 11 & \to & \end{array}$$

and each column is seen as a truth table of a 2 variable function, then the 3 functions corresponding to each column are given as y_1, y_0, y_0y_1. Similarly the corresponding functions for B^1 are $y_1, y_0, y_0y_1 \oplus 1$.

Theorem 1. *Let the function h be as described above, i.e. $h = s_0$ function of the LowMC S-box. For any guess $G = [g_{t-1}, g_{t-2}, \ldots, g_0]$, consider the following $3t$ Boolean Functions in $2t$ variables $y_0, y_1, \ldots, y_{2t-1}$.*

$$\boldsymbol{P}_{3i+2} = y_{2i+1},\ \boldsymbol{P}_{3i+1} = y_{2i},\ \boldsymbol{P}_{3i} = y_{2i}y_{2i+1} \oplus g_i,\ \forall\ i \in [0, t-1]$$

Then by evaluating these $3t$ functions over all the points $y_0, y_1, \ldots, y_{2t-1} \in \{0,1\}^{2t}$, one can generate all of the space B^G. In other words, the $3t$-bit vectors

$$V_j = \boldsymbol{P}_{3t-1}(v_j)||\boldsymbol{P}_{3t-2}(v_j)||\ldots||\boldsymbol{P}_0(v_j),\ \textit{for } j \in [0, 4^t - 1]$$

are all the vectors in B^G, where v_j is just the $2t$-bit binary representation of the integer j. Also there is a one-to-one correspondence between V_j and v_j.

Proof. To prove this, we only need to show that for every $V \in B^G$ there exists a unique j and therefore v_j such that $V = \mathbf{P}_{3t-1}(v_j)||\mathbf{P}_{3t-2}(v_j)||\ldots||\mathbf{P}_0(v_j) = V_j$. Let V be an arbitrary vector in B^G. V is therefore of the form $\mathbf{u}_{t-1}, \mathbf{u}_{t-2}, \ldots, \mathbf{u}_0$ where each $\mathbf{u}_i$ is any one of the four 3-bit vectors in B^{g_i}. For each $\mathbf{u}_i$, we can see that a unique 2-bit vector u_i that generates it. For example if $g_i = 0$, and $\mathbf{u}_i = [1,0,0]$, then it can be easily seen (by looking at the 4×3 matrix above) that for $u_i = [1,0]$, we have $\mathbf{u}_i = \mathbf{P}_{3i+2}(u_i)||\mathbf{P}_{3i+1}(u_i)||\mathbf{P}_{3i}(u_i)$ (by a slight abuse of notation). This can easily be verified for all other elements of B^0/B^1. So the unique v_j that generates V is given as $u_{t-1}||u_{t-2}||\ldots||u_0$ as described above. □

4.2 Evaluation over all points of B^G

Given an oracle $\mathcal{O}$ that given an n-bit input vector X, evaluates an n-variable Boolean function F over X, and returns $F(X)$, how many accesses to the oracle are necessary to evaluate the algebraic expression of F? For arbitrary functions, where we have no prior information about its properties, it is well-known that we need the evaluate F over all 2^n points in its input space. After this, it is equally well-known that we need to run the Möbius transform on the evaluations of F to generate the algebraic expression. Note that the Möbius transform is a completely linear operation which is involutive. Executing the same transform on the vector of coefficients in the algebraic expression, returns back the truth table of F.

However, if it is known apriori that the algebraic degree of the Boolean function is some $d < n$, then it is also well known that only $\binom{n}{\downarrow d} = \sum_{i=0}^{d} \binom{n}{i}$ evaluations of F are required. Indeed, the algebraic expression of a Boolean function F is written as

$$F = \bigoplus_{v \in \{0,1\}^n} a_v x^v,$$

where if $v = [v_{n-1}, v_{n-2}, \ldots, v_0]$ then x^v implies $x_{n-1}^{v_{n-1}} x_{n-2}^{v_{n-2}} \cdots x_0^{v_0}$. Then it is well known that the coefficient a_v is computed as $a_v = \oplus_{u \preceq v} F(u)$. Here $u \preceq v$, implies that $u_i \leq v_i$ for all i, which also implies that the hamming weight of u is less than or equal to that of v. Since any degree d coefficient a_v can be computed with evaluations of F at points $u \preceq v$, thus $\binom{n}{\downarrow d}$ accesses to the oracle are sufficient to compute a_v and therefore the entire algebraic expression. Thereafter, one can use the Möbius transform to automatically generate evaluation of F over all the points of its input space. This is an interesting property of degree d functions: an evaluation over only $\binom{n}{\downarrow d}$ points is sufficient to generate its evaluations over all of its input space.

The next question is as follows: given oracle access to a random n-variable Boolean function F of algebraic degree d, where $n = 3t$ is a multiple of 3. For any G, how many evaluations of F are required to evaluate F over the entire of B^G, where B^G is the set defined for the function h in the previous subsection? Certainly $\binom{n}{\downarrow d}$ evaluations are sufficient, since it allows us to evaluate F over its entire space and not just B^G.

Note that when we are enumerating B^G as explained in Theorem 1, i.e., when the j^{th} vector in B^G is generated as $V_j = \mathsf{P}_{3t-1}(v_j)||\mathsf{P}_{3t-2}(v_j)||\ldots||\mathsf{P}_0(v_j)$, and then evaluating F over these 4^t points, we get a list of 4^t evaluations of F. This can also be seen as the truth table of another Boolean function F over $2t = \frac{2n}{3}$ variables. Thus we have that

$$\mathsf{F}(y_{2t-1}, y_{2t-2}, \ldots, y_0) = F(x_{3t-1} = \mathsf{P}_{3t-1},\ x_{3t-2} = \mathsf{P}_{3t-2}, \ldots, x_0 = \mathsf{P}_0).$$

Example 1. *Let us say that F is a Boolean function over 9 bits of degree 3 given as $x_0x_1 \oplus x_2x_3 \oplus x_4x_5 \oplus x_7x_8 \oplus x_0x_3x_6$. If $G = [0,0,0]$, then from Theorem 1, we know that $\boldsymbol{P}_8 = y_5, \boldsymbol{P}_7 = y_4, \boldsymbol{P}_6 = y_5y_4,\quad \boldsymbol{P}_5 = y_3, \boldsymbol{P}_4 = y_2, \boldsymbol{P}_3 = y_3y_2,\quad \boldsymbol{P}_2 = y_1, \boldsymbol{P}_1 = y_0, \boldsymbol{P}_0 = y_1y_0$. Thus we can see that*

$$\begin{aligned}\boldsymbol{F} &= y_1y_0 \cdot y_0 \oplus y_1 \cdot y_2y_3 \oplus y_2 \cdot y_3 \oplus y_4 \cdot y_5 \oplus y_1y_0 \cdot y_3y_2 \cdot y_5y_4 \\ &= y_0y_1 \oplus y_1y_2y_3 \oplus y_2y_3 \oplus y_4y_5 \oplus y_0y_1y_2y_3y_4y_5\end{aligned}$$

Note that in this case, $\boldsymbol{F}$ is of degree $3 \cdot 2 = 6$, double that of F. However for any arbitrary choice of F, this is not always so. For the degree of $\boldsymbol{F}$ to be twice that of F, the algebraic expression of F must contain one term of the form $x_{3i_1} \cdot x_{3i_2} \cdots x_{3i_d}$.

Definition 1. Henceforth, we will call F_G *the associated function* of F (or simply F if B^G is clear from the context). Note that given any G, there is a 1-1 mapping between the $n = 3t$-bit vector $x = [x_{3t-1}, x_{3t-2}, \ldots, x_0]$ and the $2t$-bit vector $y = [y_{2t-1}, y_{2t-2}, \ldots, y_0]$, such that on B^G, we have $F(x) = \mathsf{F}(y)$ for all $x \in B^G$ and $y \in \{0,1\}^{2t}$. We have seen that this map is given by

$$x_{3i-1} = y_{2i-1},\ x_{3i-2} = y_{2i-2},\ x_{3i} = y_{2i-1} \cdot y_{2i-2} \oplus g_i,\quad \forall i \in [0, t-1] \tag{2}$$

So y is essentially a shorter description of x in B^G. Hence, we will call y *the associated vector* of x in B^G.

Since the P_i's are at most of degree 2, the above example makes clear that if F has degree d, then the degree of F can be at most $2d$. So one can try to compute the whole truth table of F, which is equivalent to evaluating F on all of B^G. Since the degree of F is bounded by $2d$, from the previous analysis we know that we need a total of $\binom{2t}{\downarrow 2d} = \binom{2n/3}{\downarrow 2d}$ evaluations of F for this purpose. We can use the same F oracle for this purpose: to evaluate F on any point $2t$-bit vector v_j, we first map it to the corresponding V_j and then query the oracle with it; the response is recorded as $\mathsf{F}(v_j)$. Now $\binom{2n/3}{\downarrow 2d} < \binom{n}{\downarrow d}$ does not always hold.

Consider $n = 21$. When $d = 5$ say, we have $\binom{21}{\downarrow 5} = 27896 > 2^{14} > \binom{14}{\downarrow 10}$. However when $d = 2$, we have $232 = \binom{21}{\downarrow 2} < \binom{14}{\downarrow 4} = 1471$. Hence translating the problem to F does not always yield minimal number of evaluations. However F has some structure, which can be exploited, as will be seen in the following example.

Example 2. *Let us say that F is a Boolean function over 12 bits of degree 2 given as $x_0x_1 \oplus x_2x_3 \oplus x_4x_5 \oplus x_7x_8 \oplus x_0x_3 \oplus x_9 \oplus x_{10}x_{11}$. If $G = [0,0,0,1]$, we know that $\boldsymbol{P}_{11} = y_7, \boldsymbol{P}_{10} = y_6, \boldsymbol{P}_9 = y_7y_6,\quad \boldsymbol{P}_8 = y_5, \boldsymbol{P}_7 = y_4, \boldsymbol{P}_6 = y_5y_4,\quad \boldsymbol{P}_5 = y_3, \boldsymbol{P}_4 = y_2, \boldsymbol{P}_3 = y_3y_2,\quad \boldsymbol{P}_2 = y_1, \boldsymbol{P}_1 = y_0, \boldsymbol{P}_0 = y_1y_0 \oplus 1$. Thus we can see that*

$$\begin{aligned}\boldsymbol{F} &= (1 \oplus y_1y_0) \cdot y_0 \oplus y_1 \cdot y_2y_3 \oplus y_2 \cdot y_3 \oplus y_4 \cdot y_5 \oplus (1 \oplus y_1y_0) \cdot y_3y_2 \oplus y_7y_6 \oplus y_7y_6 \\ &= y_0 \oplus y_0y_1 \oplus y_1y_2y_3 \oplus y_4y_5 \oplus y_0y_1y_2y_3\end{aligned}$$

Note that in this case, $\boldsymbol{F}$ is of degree 4, but has only one degree 4 term, whereas an arbitrary degree 4 Boolean function in 8-bits can have upto $\binom{8}{4} = 70$ such terms.

Although the above is a slightly extreme example of a sparse function, one can generalize the above example as follows. Note that if F is of degree d, the corresponding F certainly does not contain all the $\binom{2n/3}{2d}$ terms of degree $2d$. We have seen that only the monomials of form $x_{3i_1} \cdot x_{3i_2} \cdots x_{3i_d}$ in F lead to full degree terms in F. This means that the maximum degree terms must be clustered with respect to the variables, i.e., F can have maximum degree terms of type $y_0y_1 \cdot y_2y_3$, $y_0y_1 \cdot y_4y_5$ but not $y_0y_1y_2y_4$. Since F can have a maximum of $\binom{n/3}{d}$ terms of form $x_{3i_1} \cdot x_{3i_2} \cdots x_{3i_d}$, this is also the maximum number of degree $2d$ terms F can have.

Now we make 2 observations. First since $a_v = \oplus_{u \preceq v} F(u)$, the total number of evaluations of a function required to interpolate only the coefficient a_v are all the binary strings $u \preceq v$, the total number of which is $2^{hw(v)}$. This also tells us that to interpolate some coefficient a_{v^*} such that $v^* \preceq v$, we do not need any additional evaluations. So, for example, if we have the function evaluation at all the points needed to interpolate the coefficient of y_0y_1, we do not require additional points to interpolate the coefficients of y_0 or y_1 or the constant term, which are all sub-monomials of y_0y_1. Secondly consider all the $\binom{n/3}{d}$ possible maximum degree monomials of F. All other lower degree monomials of F must also be sub-monomials of at least one of these maximum degree monomials. To see why this is so, let there be a monomial $y_{j_1}y_{j_2} \cdots y_{j_{2d-1}}$ in F of degree $2d-1$ or less that is not a sub-monomial of any of the maximum degree monomials. Now group the integers j_i in the following manner: if two of them are of the form $2k, 2k+1$ put them in the same group or else put them in a different group. After this if we have $m \leq d$ such groups, then by definition it is a sub-monomial of one of the max degree monomials of F. Else if the number $m > d$, it must have been produced by a monomial of degree larger than d in F, which contradicts the fact that the algebraic degree of F is d.

Remark 1. The above observation does not mean that for any arbitrary F, all lower degree terms of F must be a sub-monomial of some maximum degree term present in F itself. Instead it means that all lower degree terms are sub-monomials of the $\binom{n/3}{d}$ max degree terms that could be potentially present in F. For example, the function F in **Example 2** contains y_4y_5 which is not a sub-monomial of $y_0y_1y_2y_3$. However, for F of 12 variables and degree 2, there can be $\binom{4}{2} = 6$ max degree terms in F, i.e. $y_0y_1y_2y_3$, $y_0y_1y_4y_5$, $y_0y_1y_6y_7$, $y_2y_3y_4y_5$, $y_2y_3y_6y_7$, $y_4y_5y_6y_7$. It can be seen that y_4y_5 is a sub-monomial of one of these.

The above two observations tell us that to interpolate F we only need its evaluations over points that are required to compute the coefficients of its maximum degree terms. We determine the evaluations of F are necessary to only find the coefficients of its $\binom{n/3}{d}$ maximum degree terms, in the following theorem.

Theorem 2. *Let F be a Boolean function of degree d over $n = 3t$ variables. Let F be the equivalent algebraic expression in $2t$ variables obtained by evaluating F over the set B^G for some G. The number of evaluations of F required to interpolate its complete algebraic expression is given as*

$$J(n,d) = \begin{cases} \sum_{i=0}^{d} \binom{n/3}{i} \cdot 3^i, & \forall d \leq n/3 \\ 2^{2n/3} & \textit{if } d > n/3 \end{cases}$$

It also holds that $J(n,d) \leq \binom{n}{\downarrow d}$ and $J(n,d) \leq \binom{2n/3}{\downarrow 2d}$ for all n, d.

Proof. We prove this by induction on d. Note that for $d = 0$, we only need the evaluation of F at one point 0^{2t}. For $d = 1$, F can only have the maximum terms of the form $y_{2k}y_{2k+1}$ for $k = 0$ to $t-1$. There are $\binom{t}{1} = \binom{n/3}{1}$ such terms. For the y_0y_1 term for example, along with 0^{2t} we need the 3 points $0^{2t-2}||p$, where $p = 01, 10, 11$. So for all the $\binom{n/3}{1}$ terms we need $3 \cdot \binom{n/3}{1}$ points plus the single point 0^{2t}. Thus the base case $d = 1$ is proven.

Let the assertion hold for any arbitrary $d > 1$. For $d+1$, consider the maximum degree term $y_{2i_1}y_{2i_1+1} \cdot y_{2i_2}y_{2i_2+1} \cdots y_{2i_{d+1}}y_{2i_{d+1}+1}$ We certainly need to evaluate F at the 3^{d+1}

points given by $y_{2i_t}y_{2i_t+1} = 10, 01, 11$ for each of $t \in [1, d+1]$. All other points for which one of the $y_{2i_t}y_{2i_t+1} = 00$ are already included in the calculation of $J(n,d)$ i.e. the number of points for degree d. Hence we have

$$J(n, d+1) = J(n,d) + \binom{n}{d+1} 3^{d+1} = \sum_{i=0}^{d+1} \binom{n/3}{i} \cdot 3^i.$$

This completes the first part of the proof. Note that if $d \geq n/3$, then F potentially can be of full degree and so all the $2^{2t} = 2^{2n/3}$ evaluations are necessary.

Note that all points included in the set of $J(n,d)$ have a special form. Let $L(n,d)$ be the set of binary strings of length $t = n/3$ and Hamming weight up to d. Then by a slight abuse of notation all the strings in the set $J(n,d)$ can be written as $\cup_{i=0}^{d} L(n,i) \otimes \{01, 10, 11\}^i$. For example if $1001 \in L(12,2)$, then $1001 \otimes [11, 01]$ is defined as 11 00 00 01.

The second part of the theorem can be proven thus. Note that for $d = 0$, $J(n,d) = \binom{n}{\downarrow d} = 1$ and for $d = 1$, $J(n,d) = \binom{n}{\downarrow d} = n+1$. For larger d, we have the i^{th} term of α_i of $J(n,d)$ is

$$\alpha_i = \binom{n/3}{i} 3^i = \frac{(n/3)\cdots(n/3-i+1)}{i!} \cdot 3^i = \frac{(n)\cdot(n-3)\cdots(n-3i+3)}{i!}$$

On the other hand the i^{th} term of β_i of $\binom{n}{\downarrow d}$ is $\beta_i = \binom{n}{i} = \frac{(n)\cdot(n-1)\cdots(n-i+1)}{i!}$ By inspecting α_i and β_i term by term it is clear that $\alpha_i < \beta_i$ and so $J(n,d) \leq \binom{n}{\downarrow d}$ follows. For $d > n/3$, $J(n,d)$ becomes constant whereas $\binom{n}{\downarrow d}$ continues to increase and so the inequality holds for $d > n/3$ too. For the second inequality, note that for $d = 0$ we have $J(n,d) = \binom{2n/3}{\downarrow 2d} = 1$, and for $d \geq n/3$ we have $J(n,d) = \binom{2n/3}{\downarrow 2d} = 2^{2n/3}$. For other values of d, instead of a mathematical proof, this inequality can be understood intuitively: $\binom{2n/3}{\downarrow 2d}$ is the total number of binary strings of length $2n/3$ and Hamming weight up to $2d$. Whereas we have just shown by construction that $J(n,d)$ is just the size of a small subset of all such strings of length $2n/3$ and Hamming weight up to $2d$.

□

4.3 Efficient Algorithms for Evaluation

There are two tasks we need to consider here: first given the evaluation on $J(n,d)$ points, how to evaluate the algebraic normal form (i.e., vector of coefficients of algebraic expression) of F, and second given the algebraic normal form of F how to evaluate the truth table on all its points. Note that if we had the evaluation of all the points in the input space of F, then both the above tasks can be achieved by a simple in place execution of the Möbius transform that requires $2^{2n/3}$ bits of space and $O(n \cdot 2^{2n/3})$ bit operations.

The first algorithm requires that the attacker be able to map **(a)** each of the $J(n,d)$ vectors $v \in \{0,1\}^{2t}$ that is into an index $j \in [0, J(n,d)-1]$ and **(b)** an operation that computes the inverse map efficiently. Thereafter, each evaluation $\mathsf{F}(v)$ is stored in the array location j. After this we can apply an algorithm similar to one iteration of the standard Möbius transform a total of $2t = 2n/3$ times. We prove in Appendix A, that this takes around $\frac{2n}{3} \cdot J'(n,d)$ bit-operations, where $J'(n,d) = 2 \cdot \sum_{i=0}^{d-1} \binom{n/3-1}{i} \cdot 3^i$. The total space required in this algorithm apart from the $J(n,d)$ bits required to store the evaluation array are a few pre-computed tables to speedup the routine. In Appendix A, we prove that the additional space is bounded by $\left(\frac{2n}{3} \cdot \log_2 J(n,d)\right) \cdot J(n,d)$ bits.

The second algorithm is the efficient Möbius transform that was already proposed in [DS11, Sec 3.2]. Specifically, the co-efficients of the algebraic normal form are redistributed into an array of length $2^{2n/3}$. Thereafter, at the i-th step ($0 \leq i < 2n/3$), the array is divided into 2^{i+1} sub-arrays and only the indices whose hamming weight is $\leq 2d$ in the least significant $g = 2n/3 - i - 1$ bits in one-half of the sub-arrays are updated. In Appendix A, it is shown that the total time complexity of this step is around $O(2d \cdot 2^{2n/3})$ xor operations.

In Appendix A, we further present complete algorithms for both subroutines. Note that if T_{oracle} is the time required to evaluate F at one point then the above operation requires $J(n, d) \cdot T_{oracle}$ bit operations to generate the evaluations. Furthermore, the two routines to generate the algebraic expression for F and then its truth table takes

$$T(n, d) = \frac{2n}{3} \cdot J'(n, d) + 2d \cdot 2^{2n/3} \text{ bit operations.} \tag{3}$$

The total space required is around

$$M(n, d) = \frac{2n}{3} \cdot J(n, d) \cdot \log_2 J(n, d) + 2^{2n/3} \approx 2^{2n/3} \text{ bits.} \tag{4}$$

4.4 Finding F on B^G and $B^{G'}$ when $\mathbf{hw}(G \oplus G') = 1$

Given two guess vectors with G and G' with Hamming difference one, i.e., $\mathrm{hw}(G \oplus G') = 1$, we can observe that there is some similarity between F_G and $\mathsf{F}_{G'}$. Without loss of generality, let $G = b_0, b_1, \ldots, b_{t-1}$ and $G' = 1 \oplus b_0, b_1, \ldots, b_{t-1}$, where all $b_i \in \{0, 1\}$. Let $\overline{F}$ be the derivative of F over the coordinate x_0, i.e. $\overline{F} = F(\ldots, x_0) \oplus F(\ldots, x_0 \oplus 1)$. Consider the associated function of $\overline{\mathsf{F}}_G$ of $\overline{F}$. We have

$$\begin{aligned}
\overline{\mathsf{F}}_G &= \overline{F}(\ldots, y_1, y_0, y_0 y_1 \oplus b_0) \\
&= F(\ldots, y_1, y_0, y_0 y_1 \oplus b_0) \oplus F(\ldots, y_1, y_0, y_0 y_1 \oplus b_0 \oplus 1) \\
&= \mathsf{F}_G \oplus \mathsf{F}_{G'}.
\end{aligned}$$

Thus the difference between F_G and $\mathsf{F}_{G'}$, is equal to the associated function of the derivative $\overline{F}$ on B^G. Since $\overline{F}$ is a derivative it is of degree $d - 1$, and hence $\overline{\mathsf{F}}_G$ is of degree $2d - 2$. Since it is an associated function, its algebraic expression has the same sparse structure. Let us say we have already the truth table for F_G. When trying to evaluate F in the set $B^{G'}$, we can only interpolate up to the degree $2d - 2$ terms of $\overline{\mathsf{F}}_G$. This reduces the number of evaluations to $J(n, d - 1)$ in place of $J(n, d)$.

In practice, in order to evaluate $\overline{\mathsf{F}}_G$, we actually need evaluations of both $F(\ldots, x_0)$ and $F(\ldots, x_0 \oplus 1)$ over $J(n, d - 1)$ points. The former are the evaluations of F on B^G which are already stored in the truth table of F_G. The latter are the evaluations of F on $B^{G'}$ which we additionally need. Thereafter the difference of the evaluations on these set of $J(n, d - 1)$ points is used to interpolate the algebraic expression and thereafter the truth table of $\overline{\mathsf{F}}_G$. We then find $\mathsf{F}_{G'} = \mathsf{F}_G \oplus \overline{\mathsf{F}}_G$. This does not require any additional memory except the space required to store the truth tables of successive F_G's. Thus when we need the truth tables of F over multiple partial sets B^G, if possible it is more efficient to traverse the guess space in a Gray code like manner, in which each successive guess vector has a Hamming distance 1 from the immediately previous guess vector. This way, each successive evaluation takes $J(n, d - 1)$ points. The algorithm is explained formally in Algorithm 1.

4.5 Cube sum over partial space

We look at one final result connected with partial sets before moving on to the attack description. Let F be any Boolean function over n variables of degree d. Partition the n variables $x_0, x_1, \ldots, x_{n-1}$ into 2 sets $X_1 = [x_0, x_1, \ldots, x_{n_1-1}]$ and $X_2 = [x_{n_1}, x_{n_1+1}, \ldots, x_{n-1}]$ of size n_1 and $n - n_1$ respectively. We know that the Boolean function $F' = \bigoplus_{X_1 \in \{0,1\}^{n_1}} F(X_2, X_1)$

Algorithm 1: Evaluation of $\mathsf{F}_{G'}$ from F_G assuming $\mathrm{hw}(G \oplus G') = 1$.

Input: Number of variables n, degree of $F = d$, iteration number i
Input: Truth table for F_G if $i \neq 0$
Output: Truth table for $\mathsf{F}_{G'}$ if $i \neq 0$ else F_G
if *i=0* **then**
  /* `First Iteration`*/
  Get $J(n,d)$ evaluations of F on B^G
  Interpolate expression F_G using $\text{Möbius}_{\mathbf{2}}(2n/3)$ in Appendix A;
  Evaluate truth table F_G using $\text{Möbius}_{\mathbf{3}}(2n/3, 2d)$ in Appendix A;
  Store truth table in array `Tab`;
end
else
  Get $J(n, d-1)$ evaluations of F on $B^{G'}$;
  /* `Equivalent to evaluations of` $\mathsf{F}_{G'}$ `on` all $y \in \{0,1\}^{2n/3}$*/
  for *Each* $x \in$ *set of* $J(n, d-1)$ **do**
    Find associated vector y of x;
    $\overline{\mathsf{F}}_G(y) = \mathsf{Tab}(y) \oplus \mathsf{F}_{G'}(y)$;
  end
  Interpolate expression $\overline{\mathsf{F}}_G$ using $\text{Möbius}_{\mathbf{2}}(2n/3)$ in Appendix A;
  Evaluate truth table $\overline{\mathsf{F}}_G$ using $\text{Möbius}_{\mathbf{3}}(2n/3, 2d-2)$ in Appendix A;
  $\mathsf{Tab}(y) = \mathsf{Tab}(y) \oplus \overline{\mathsf{F}}_G(y), \ \forall\, y$; /* `Truth-table of` $\mathsf{F}_{G'}$ `is in Tab`*/
end

which is a cube sum over the cube represented by n_1 bits, is a function of degree at most $d - n_1$ over $n - n_1$ variables.

Now consider a function F over $n = 3t$ variables of degree d, and the function h for which we defined the set B^0/B^1 for a guess vector in the previous sub-sections. Again partition the $n = 3t$ variables $x_0, x_1, \ldots, x_{3t-1}$ into 2 sets $X_1 = [x_0, x_1, \ldots, x_{3t_1-1}]$ and $X_2 = [x_{3t_1}, x_{3t_1+1}, \ldots, x_{3t-1}]$ of size $n_1 = 3t_1$ and $n - n_1 = 3t - 3t_1$ respectively. Now for some guess vector $G_1 \in \{0,1\}^{t_1}$ define the set $B^{G_1} = B^{g_{t_1-1}} \times B^{g_{t_1-2}} \times \cdots \times B^{g_0}$ of size 2^{2t_1}. Now consider the Boolean function F'' defined as $F'' = \bigoplus_{X_1 \in B^{G_1}} F(X_2, X_1)$. We will try to determine algebraic degree of F''.

Note that the function h for which $B^0 = \{0, 2, 4, 7\}$ has the algebraic expression $x_0 \oplus x_1 x_2$. Let H be any 3-variable Boolean function: the sum of H over the set B^0 can be clearly seen as $\bigoplus_{x \in \{0,1\}^3} H(x) \cdot (1 \oplus h(x))$ and that over B^1 is $\bigoplus_{x \in \{0,1\}^3} H(x) \cdot h(x)$. Therefore we have

$$F'' = \bigoplus_{X_1 \in B^{G_1}} F(X_1, X_2) = \bigoplus_{X_1 \in \{0,1\}^{3t_1}} F(X_1, X_2) \cdot \prod_{i=0}^{t_1-1} (g_i \oplus 1 \oplus h(x_{3i}, x_{3i+1}, x_{3i+2}))$$

Since h is quadratic, this is a cube sum over $3t_1$ bits of a function of degree $d + 2t_1$, and so its degree is at most $d + 2t_1 - 3t_1 = d - t_1 = d - n_1/3$.

5 Details of the attack

After linearizing the first round, the attacker can obtain n equations in the n-keybit variables of degree $d = 2^\rho$ each for any arbitrary instance of $2\rho/2\rho+1$ round LowMC. Let us denote the equations $E_{G,i} = 0, \ \forall\, i \in [0, n-1]$, where the suffix G denotes the guess vector used to linearize and construct the equations. The central technique of equation solving popularized in [LPT+17, Din21a, Din21b] is formulating the Boolean polynomial $A_G = \prod_{i=0}^{n-1}(1 + E_{G,i})$ in the keybit variables. Note that if and only if $K^* \in B^G$ is a common root of all the $E_{G,i}$, then A_G evaluates to 1 at the point K^* (for convenience we will call all points that evaluate to 1 with A_G as its solution space). Note that if the $E_{G,i}$'s have a unique/odd number of roots in B^G then the sum $S = \bigoplus_{x \in B^G} A_G(x)$ will return 1. S therefore serves as a decision oracle that returns if an underlying equation system has a unique or odd number of roots. If on the other hand the given equation system does have a unique root, and if given $E_{G,i}$'s one can efficiently compute S, then it was shown in [LPT+17], how to query this oracle a polynomial number of times to recover the unique root.

However each $E_{G,i}$ has potentially $\binom{n}{\downarrow d}$ terms and multiplying n such equations to get A_G is computationally expensive and is unlikely to take time less than exhaustive search of key, at least for the parameter sets we are interested in. So a little improvisation is required to compute the polynomial efficiently. Note in all instances of LowMC with complete non-linear layers the blocksize/keysize $n = 3t$ is a multiple of 3. So We first partition the key variables $k_{3t-1}, k_{3t-2}, \ldots, k_0$ into two sets $K_2 = [k_{3t-1}, k_{3t-2}, \ldots, k_{3t_1}]$ and $K_1 = [k_{3t_1-1}, k_{3t_1-2}, \ldots, k_0]$ of size $n - n_1 = 3t - 3t_1$ and $n_1 = 3t_1$ each. Since we have used the function h to linearize the first round, **(a)** we need to repeat the root finding process a total of 2^t times, once for each $G \in \{0,1\}^t$, and **(b)** for any specific G we limit all the arithmetic in the set B^G instead of the whole of $\{0,1\}^n$, since we are only interested to find roots in B^G.

As a result of partitioning the key variables into X_2, X_1, this induces the natural partition of the bits of $G = G_2||G_1$, where $G_1 = [g_{t_1-1}, g_{t_1-2}, \ldots, g_0]$ and $G_2 = [g_{t-1}, g_{t-2}, \ldots, g_{t_1}]$. The next idea is for some key vector in B^{G_2}, we perform an exhaustive search in B^{G_1}. First we choose a parameter $\ell < n$. Next we randomly choose ℓ out of the n equations and try to find the common roots of these ℓ equations. Note this induces an underlying random polynomial $\tilde{A}_G = \prod_{i=0}^{\ell-1}\left(1 \oplus E_{G,r(i)}\right)$, where $r(i)$ is the i^{th} element in the list of ℓ random integers chosen in $[0, n-1]$. $\tilde{A}_G$ evaluates to 1 only at the common roots of the ℓ random equations chosen above. As such $\tilde{A}_G$ is essentially a noisy version of A_G that is slightly easier to compute. Before we proceed further let us look at the following lemma.

Lemma 4. *Given a single plaintext and ciphertext produced by a LowMC instance, using a key $K^* \in B^{G^*}$ for some G^*. After linearizing with the guess vector $G \in \{0,1\}^t$, let $E_{G,i}$, for $i \in [0, n-1]$ be the n equations, so obtained. Let the product polynomials A_G, $\tilde{A}_G$ be constructed as defined above after making a random selection of ℓ equations. Then under the assumption that, for any G, half the points in B^G take $E_{G,i}$ to 0 and the other half to 1, we have*

a) *For all $G \neq G^*$, $\boldsymbol{Pr}[\tilde{A}_G(K) = 1] \approx 2^{-\ell}$, $\forall\ K \in B_G$.*

b) *For $G = G^*$, $\tilde{A}_G(K^*)$ has to be 1 by construction for any choice of ℓ equations. For all other K, we again have $\boldsymbol{Pr}[\tilde{A}_G(K) = 1] \approx 2^{-\ell}$.*

Note that all the above probabilities are computed for all random choices of ℓ of the n equations.

Proof. Under the theorem assumption, the probability that any $K \in B_G (\neq B_{G^*})$ is a root of any single $E_{G,i}$ can be considered to be around $\frac{1}{2}$. Under the assumption of independence, the probability that any $K \in B_G$ is a common root of ℓ equations is thus approximately $\left(\frac{1}{2}\right)^{\ell} = 2^{-\ell}$. This argument can also be extended to all points $K \neq K^*$, when the guess vector $G = G^*$ is correct. Since K^* by construction has to be the common root of all $E_{G^*,i}$, we must have $\tilde{A}_{G^*}(K^*) = 1$, for all random choices of ℓ equations. This also tells us that the expected solution space of $\tilde{A}_G$ in the set B^G, for all choices of G, has cardinality around $2^{2n/3-\ell}$.

Note that we assume that $E_{G,i}$'s are balanced and independent in B^G to arrive at the proof. It is a reasonable assumption to make for large values of n. We verified by computer simulations for smaller LowMC instances with blocksize up to 24, that the $E_{G,i}$'s are close to balanced and independent on the partial sets.

□

Now the main idea is as follows: we fix some constant $u \in B^{G_2}$, and try to find all common roots of the reduced equation system $E_{G,r(i)}(u, K_1)$, for $i \in [0, \ell - 1]$. This is an equation system in only $n_1 = 3t_1$ variables, and therefore we can exhaustively search for common roots of this reduced system.

5.0.1 Data Generation

In this part we describe how the attacker collects data to proceed with the attack. The first step is obviously to find a truth table for $\tilde{A}_G(u, K_1)$ for some fixed $u \in B^{G_2}$. We proceed as follows.

a) We will not compute the algebraic expression for any equation $E_{G,r(i)}$ for any G, i. Instead what we do is as follows. Choose any $u \in B^{G_2}$. For all $v \in B^{G_1}$, for the vector $u||v$ we need to evaluate $E_{G,r(i)}(u, v)$. There are $2^{2n_1/3}$ of such points.

b) For each of the $2^{2n_1/3}$ points $(u, v) = k$ (say), consider k to be the LowMC key. If R is odd, execute the first $\rho + 1$ rounds with k as key with the given plaintext to get the state SF. Then execute the inverse of last ρ rounds with k as key on the given ciphertext, to get the state SB. If R is even, then execute the first ρ rounds and the Substitution layer of the $\rho + 1$-th round to get S_F (see Fig 2). Then execute the inverse of the last $\rho - 1$ rounds and the inverse round key addition and affine layer of round $\rho + 1$ to get S_B. If the $r(i)$-th bits SF and SB are equal, then k is obviously a root of $E_{G,r(i)}$ i.e. $E_{G,r(i)}(u, v) = 0$. If not we have $E_{G,r(i)}(u, v) = 1$. Note that this also allows us to evaluate $E_{G,i}(u, v)$ for all $i \in [0, n - 1]$ by simply checking whether i-th bits SF and SB are equal.

c) (u, v) is a common root of the system of equations $E_{G,r(i)}(u, v)$, for $i \in [0, \ell - 1]$ if $r(i)$-th bits SF and SB are equal for all i.

d) The method requires $2 \cdot (2\rho + 1)n^2 = 2Rn^2$ operations for each point and so the total number of bit operations required for this operation for any one $u \in B^{G_2}$ is $2Rn^2 \cdot 2^{2n_1/3}$.

In fact, finding the common roots of $E_{G,r(i)}(u, K_1)$ is equivalent to finding the truth-table of the n_1-variable Boolean polynomial $\tilde{A}_G(u, K_1)$ over the points in B^{G_1}. This is true since $\tilde{A}_G(u, K_1)$ evaluates to 1 only at these common roots and 0 otherwise. Define the polynomials $F_G = \bigoplus_{v \in B^{G_1}} A_G(K_2, v)$ and $\tilde{F}_G = \bigoplus_{v \in B^{G_1}} \tilde{A}_G(K_2, v)$, both of which are of $n - n_1$ variables, and by the arguments in Section 4.5, $\tilde{F}_G$ has an algebraic degree at most $d\ell - n_1/3$. If we now find the sum of all the points in the truth table of $\tilde{A}_G(u, K_1)$ over B^{G_1} that we have just found, we will be essentially evaluating $\tilde{F}_G$ at the point $u \in B^{G_2}$.

Remark 2. Note that we have seen that the number of operations needed to evaluate $\tilde{F}_G$ at the single point $u \in B^{G_2}$ is $T_{oracle} = 2Rn^2 \cdot 2^{2n_1/3}$. And from the analysis in Section 4.2, we know that we need evaluations of $\tilde{F}_G$ at $J(n - n_1, d\ell - n_1/3)$ points u to fully find its truth table over B^{G_2}. Note that, as we will see later, since the method is probabilistic, we will need to perform this operation multiple (say N) times: each time for a different combination of ℓ equations $E_{G,i}$ in $[0, n - 1]$. Whereas we have seen that performing the steps **a-d** allows us to evaluate $E_{G,i}(u, v)$ for all $i \in [0, n - 1]$ for any u and all $v \in B^{G_1}$. Note that for each u, if we store $E_{G,i}(u, v)$ in a table (for all $i \in [0, n - 1]$, $v \in B^{G_1}$), this saves us the trouble of having to re-evaluate these values when we repeat the process for a different set of ℓ equations in $[0, n-1]$. The total memory required for this will be

$$M_{eval} = J(n - n_1, d\ell - n_1/3) \cdot n \cdot 2^{2n_1/3} \text{ bits.} \tag{5}$$

Since the process is needed to be done once for the $J(n - n_1, dl - n_1/3)$ points, the time required is given as

$$T_{eval} = J(n - n_1, d\ell - n_1/3) \cdot T_{oracle}. \tag{6}$$

Just as $\tilde{A}_G$ is a noisy version of A_G, $\tilde{F}_G$ is a noisy version of F_G. We now make 2 observations. Note that for the correct guess $G = G^* = G_2^*||G_1^*$, and the correct root $K^* = K_2^*||K_1^*$, we always have $F_{G^*}(K_2^*) = 1$, since of the $2^{2n_1/3}$ terms $A_{G^*}(K_2^*, v)$ we use to construct this sum, the term evaluates to 1 only when $v = K_1^*$ (assuming that there is a unique solution). Similarly (under the same unique root assumption) we will have $F_G(u) = 0$, for all a) $G \neq G^*$ and b) $G = G^*$ and $u \neq K_2^*$. This follows since: A_G evaluates to 1 only at $G = G^*, K = K^*$, at all other points it will evaluate to 0, and so the expression for $F_G(u)$ for the above 2 cases simply sums evaluations of A_G at which it is always 0.

In [Din21a], it was proven that $\tilde{F}_{G^*}(K_2^*)$ also is 1 with high probability if ℓ is properly chosen. In fact, [Din21a] had proven that if $z = z_2||z_1$ is an *isolated solution* of a complete equation system identified by the product polynomial A with respect to the given partition of bits, (which means that z is a root and any other $z_2||z_1'$ for $z_1 \neq z_1'$ is *not* a root) then z is also an isolated solution of the noisy equation system $\tilde{A}$, with high probability, provided ℓ is chosen judiciously.

5.0.2 Evaluating $\tilde{F}_G$:

Note that $\tilde{F}_G$ has maximum algebraic degree $d\ell - n_1/3$. So $\tilde{F}_G$ can be interpolated using the set of $J(n - n_1, d\ell - n_1/3)$ evaluations of $\tilde{A}_G$. After this we use the Möbius transform algorithm described in Sec 4.3 to evaluate its truth table. This takes time and memory proportional to $T(n - n_1, d\ell - n_1/3)$ and $M(n - n_1, d\ell - n_1/3)$ for any random choice of ℓ equations. We have already seen that since this algorithm of is probabilistic, for every guess vector G, we may need to repeat it N times (for some integer N) to obtain the correct solution with high probability. Hence the complexities need to be multiplied by N.

However if we went about traversing the guess space in Gray code like manner, i.e. in the i-th step the Guess vector is $G^i = i \oplus (i \gg 1)$, then we have already seen in Section 4.4 and Algorithm 1, that $\tilde{F}_{G^{i+1}}$ can be computed more efficiently from the knowledge of $\tilde{F}_{G^i}$. We know that each additional step takes

1. Only around $J(n - n_1, d\ell - n_1/3 - 1)$ points to evaluate and hence T_{eval} becomes $J(n - n_1, d\ell - n_1/3 - 1) \cdot T_{oracle}$ for every successive guess vector. Hence over all the $2^{n/3}$ guess vectors we have

$$\begin{aligned} T_{eval,total} &= T_{oracle} \cdot (J(n - n_1, d\ell - n_1/3) + (2^{n/3} - 1) \cdot J(n - n_1, d\ell - n_1/3 - 1)) \\ &\approx 2^{n/3} \cdot J(n - n_1, d\ell - n_1/3 - 1) \end{aligned} \tag{7}$$

2. Time proportional to $T(n - n_1, d\ell - n_1/3 - 1)$ by using Algorithm 1. Thus the total time required to evaluate the truth tables over all the $2^{n/3}$ guess spaces is

$$\begin{aligned} T_{int} &= N \cdot \Big(T(n - n_1, d\ell - n_1/3) + (2^{n/3} - 1) \cdot T(n - n_1, d\ell - n_1/3 - 1)\Big) \\ &\approx N \cdot 2^{n/3} \cdot T(n - n_1, d\ell - n_1/3 - 1) \end{aligned} \tag{8}$$

To take advantage of this reduction we have to use the same set of ℓ random equations $E_{G,i}$ (for each of the N instances) over all the guess vectors G. Therefore the idea is to populate a set of N random lists r_j (for $j = 0 \to N - 1$) of length ℓ and each with integers from 0 to $n-1$. For each guess vector G, we repeat compute $\tilde{A}_G, \tilde{F}_G$ for each of these fixed lists r_j. Thus for each list r_j and each successive guess vector G^{i+1} in the Gray code order, this helps us compute $\tilde{F}_{G^{i+1}}$ using lesser evaluations of $\tilde{A}_{G^{i+1}}$ and lesser time complexity using the already computed $\tilde{F}_{G^i}$ as per Algorithm 1. As we have seen this does not require additional memory but we do have to store the N truth tables for each $\tilde{F}_G$, which takes $M_{store} = N \cdot 2^{(2n-2n_1)/3}$ bits of memory.

5.0.3 Calculating probabilities and total complexity:

We have already seen that if we assume that the underlying LowMC encryption algorithm admits a unique solution, then both $F_G(u)$, $\forall u \in B^{G_2}$, $G \neq G^*$ and $F_{G^*}(u)$ $\forall u \in B^{G_2} - \{K_2^*\}$ should be 0. We will try to find the corresponding values for $\tilde{F}_G$ in the following lemma.

Lemma 5. *Let F_G and $\tilde{F}_G$ be as defined above. Assuming that the underlying LowMC encryption admits a unique root $K^* = K_2^*||K_1^* \in B^{G^*}$, then we have for all $G = G_2||G_1$,*

$$\mathbf{Pr}[\tilde{F}_G(u) = F_G(u)] \approx 1 - 2^{2n_1/3-\ell}, \quad \forall\ u \in B^{G_2}.$$

Note that all the above probabilities are computed for all random choices of ℓ of the n equations.

Proof. We know that $\tilde{F}_G(u) = \bigoplus_{v \in B^{G_1}} \tilde{A}_G(u, v)$. Note that under the assumption of unique root, we have $A_{G^*}(K_2^*, K_1^*) = 1$ and for all other combinations of G, u, v we have $A_G(u, v) = 0$. Lemma 4 essentially therefore tells us that $\mathbf{Pr}[\tilde{A}_G(K_2^*, K_1^*) = A_G(K_2^*, K_1^*)] = 1$ and $\mathbf{Pr}[\tilde{A}_G(u, v) = A_G(u, v)] \approx 1 - 2^{-\ell}$ for all other G, u, v. Now we have

$$\begin{aligned}\mathbf{Pr}[\tilde{F}_G(u) = F_G(u)] &= \mathbf{Pr}\left[\bigoplus_{v \in B^{G_1}} \tilde{A}_G(u, v) = \bigoplus_{v \in B^{G_1}} A_G(u, v)\right] \\ &= \mathbf{Pr}\left[\bigoplus_{v \in B^{G_1}} \tilde{A}_G(u, v) \oplus A_G(u, v) = 0\right]\end{aligned}$$

For $G \neq G^*$ or $G = G^*, u \neq K_2^*$, this is the probability that $2^{2n_1/3}$ noisy bits sum to 0, where each bit is 0 with probability $1 - 2^{-\ell}$. Assuming that the bits are i.i.d we can use the piling-up lemma to arrive at the given result. For $G = G^*, u = K_2^*$, this the probability that $2^{2n_1/3} - 1$ noisy bits sum to 0, where each bit is 0 with probability $1 - 2^{-\ell}$. Again using the piling-up lemma we arrive at the given result. This implies that if we take $2n_1/3 = \ell - 1$, then the corresponding probability is around $\frac{1}{2}$.

□

Corollary 1. *The above lemma essentially proves that, we have $\mathbf{Pr}[\tilde{F}_{G^*}(K_2^*) = 1] = 1 - 2^{2n_1/3-\ell}$. And similarly for $G \neq G^*$ or $G = G^*, u \neq K_2^*$ we have $\mathbf{Pr}[\tilde{F}_G(u) = 1] = 2^{2n_1/3-\ell}$.*

So the plan would be to evaluate $\tilde{F}_G$ at all points $u \in B^{G_2}$ or equivalently $\tilde{\mathbf{F}}_G$ over all of $\{0,1\}^{(2n-2n_1)/3}$. With some probability we can therefore uncover the first $n - n_1$ bits of the root i.e., K_2^* as the value that takes $\tilde{F}_G$ to 1 for the correct G. The overheads that the attacker must factor in are the following:

1. The attacker does not know the correct value of G^* apriori.

2. Even for the correct G^*, K_2^*, $\tilde{F}_G(K_2^*)$ may not necessarily evaluate to 1.

3. For both the cases **(a)** $G = G^*, u \neq K_2^*$, and **(b)** $G \neq G^*$, we may get $\tilde{F}_G(u) = 1$.

The attacker needs to repeat the procedure multiple times (let's say N times), each time for a new set of ℓ random equations, in order to filter out the root by some probabilistic analysis. We have already seen in Remark 2 that this does not cost additionally in terms of evaluations to generate the polynomials $\tilde{F}_G$ each time we need to select ℓ random equations. If $\tilde{F}_G^{(i)}(u)$ is the value of $\tilde{F}_G(u)$ obtained at the i^{th} such step, consider the integer sum $C_G(u) = \sum_{i=1}^{N} \tilde{F}_G^{(i)}(u)$. Naturally C_G acts essentially as a counter and is a map from $B^{G_2} \to \mathbb{Z}$.

Lemma 6. *The expected value of $C_G(u)$ for* **(a)** $G = G^*, u \neq K_2^*$, *and* **(b)** $G \neq G^*$ *is* $N \cdot 2^{2n_1/3-\ell}$. *However the expected value of $C_{G^*}(K_2^*)$ is $N \cdot (1 - 2^{2n_1/3-\ell})$*

Proof. Since C_G is an integer sum, from the observation in Corollary 1, it can be seen that $C_{G^*}(K_2^*) \sim$ *Binomial* $(N, 1 - 2^{2n_1/3-\ell})$ and for all other G, u we have $C_G(u) \sim$ *Binomial*$(N, 2^{2n_1/3-\ell})$ and so the result follows. □

The idea is to run the algorithm a total of N times for each G so that the 2nd error probability (that of an incorrect key being identified as correct) is a sufficiently small value, let's say ϵ_2. This way for each G, around $\epsilon_2 \cdot 2^{2(n-n_1)/3}$ incorrect candidates are identified on average. We can test these incorrect candidates, by running the encryption algorithm. We do this by fixing a threshold $\theta < N$, and rejecting a solution u if $C_G(u) \leq \theta$. The error probability may then be calculated as

$$\epsilon_2 = \sum_{i=\theta+1}^{N} \binom{N}{i} \cdot p^i q^{N-i}, \tag{9}$$

where $p = 2^{2n_1/3-\ell}$ and $q = 1 - p$. Once we fix a θ, the success probability is given as the probability that $C_{G^*}(K_2^*) > \theta$, and is therefore given as

$$\epsilon_1 = \sum_{i=\theta+1}^{N} \binom{N}{i} \cdot q^i p^{N-i}. \tag{10}$$

The full algorithm in the form of a subroutine is presented in Algorithm 2.

5.1 Time and Space Complexity

5.1.1 Odd rounds:

For an odd number of rounds $R = 2\rho + 1$, we have $d = 2^\rho$. The attack needs to be repeated $2^{n/3}$ times once for each guess of G, however it is noteworthy that the memory complexity remains $M(n - n_1, d\ell - n_1/3) + M_{eval}$, where $M(n, d)$, M_{eval} are as defined in Equations (4) and (5) respectively. Add to that $\log_2 N \cdot 2^{(2n-2n_1)/3}$ bits to store the counters $C_G(u)$ for each u. So the total memory complexity in terms of number of bits is

$$MC = \log_2 N \cdot 2^{(2n-2n_1)/3} + M(n - n_1, d\ell - n_1/3) + M_{eval} + M_{store}. \tag{11}$$

For each G, we get around $\epsilon_2 \cdot 2^{2(n-n_1)/3}$ candidates to test for correctness on average for the first $n - n_1$ bits of the key. Appending the $2^{2n_1/3}$ candidates in B^{G_1}, we get $\epsilon_2 \cdot 2^{2n/3}$ candidates for each guess of G. Thus testing solutions requires around $T_{test} = \epsilon_2 \cdot 2^{(2n)/3} \cdot (2Rn^2)$ bit operations for each guess of G. The total time complexity in terms of number of bit operations is given as

$$TC = T_{eval,total} + T_{int} + 2^{n/3} \cdot T_{test}, \tag{12}$$

where $T_{eval,total}$, T_{int} are as defined in Equations (7),(8). For $n = 255, R = 5$, we have $d = 4$ after linearization. If we choose $n_1 = 12$, $\ell = 2n_1/3 + 3$, and $N = 12$ and $\theta = 9$,

Algorithm 2: The algorithm for solving for the key.

Input: (pt, ct), n_1: Internal partition, ℓ: #Equations to construct $\tilde{A}_G$
Input: N: #Instances algorithm per guess of G, R: #LowMC rounds.
Input: θ: Counter Threshold
Output: The key K^* such that $Enc_{K^*}(pt) = ct$

for $j = 0 \to N-1$ **do**
    Populate array $r_j(\cdot)$ with ℓ random integers from $[0, n-1]$;
    `/* For` R `even, the random list is chosen as per Sec 5.1 */`
end
Set $i \leftarrow 0$;
for *Each guess vector* $G = G_2 || G_1 = i \oplus (i \gg 1)$ **do**
    if $i = 0$ **then**
        Set $d' = d\ell - n_1/3$ **else** $d' = d\ell - n_1/3 - 1$;
    end
    for *Each of the* $J(n - n_1, d')$ *points* $u \in B^{G_2}$ **do**
        for *Each of the* $2^{2n_1/3}$ *points* $v \in B^{G_1}$ **do**
            Compute $E_{G,i}(u, v)$ as explained in Remark 2 and store in table;
        end
    end
    `/* Evaluation complete */`
    for $j = 0 \to N-1$ **do**
        for *Each of the* $J(n - n_1, d')$ *points* $u \in B^{G_2}$ **do**
            $\tilde{F}_G(u) \leftarrow 0$;
            for *Each of the* $2^{2n_1/3}$ *points* $v \in B^{G_1}$ **do**
                $\tilde{A}_G(u, v) \leftarrow 1$ if all $E_{G, r_j(t)}(u, v) = 0$ else $\tilde{A}_G(u, v) \leftarrow 0$;
                $\tilde{F}_G(u) \leftarrow \tilde{F}_G(u) \oplus \tilde{A}_G(u, v)$;
            end
        end
        Evaluate $\tilde{F}_G$ on all points in B^{G_2};
        `/* Use the Algorithm 1 for this purpose*/`
        $C_G(u) \leftarrow C_G(u) + \tilde{F}_G(u), \quad \forall u \in B^{G_2}$
    end
    `/* Now testing of solutions begins */`
    for *Each* $u \in B^{G_2}$ **do**
        if $C_G(u) > \theta$ **then**
            for *Each of the* $2^{2n_1/3}$ *points* $v \in B^{G_1}$ **do**
                $k \leftarrow u || v$;
                if $Enc_k(pt) = ct$ **then**
                    return k;
                end
            end
        end
    end
    Update $i \leftarrow i + 1$;
end

we get $\epsilon_2 = 2^{-24}$. We get $TC \approx 2^{257}$ and $MC \approx 2^{166}$ bits and a success probability of $\epsilon_1 \approx 0.82$. The brute force complexity for the same is around 2^{274} bit-operations. Note that [Din21a] had reported $TC = 2^{251}$ operations (however we show that this complexity of around $2^{254.4}$ in Sec 6.1) with $MC = 2^{228}$ bits with success probability $\frac{11}{16} \approx 0.68$. The attack depended on a probability $\frac{1}{2}$ event occurring at least twice in 4 iterations (this is the probability that probabilistic equation system admits an isolated root which turned out to be 0.5 for the values of n_1, ℓ chosen in the paper). The probability of this is given as $\sum_{i=2}^{4} \binom{4}{i} 2^{-4} = \frac{11}{16}$.

For $n = 255, R = 7$, we have $d = 8$ after linearization. If we choose $n_1 = 6,\ \ell = 2n_1/3+2$, and $N = 17$ and $\theta = 12$, we get $TC \approx 2^{262}$ and $MC \approx 2^{170}$ bits and a success probability of $\epsilon_1 \approx 0.57$.

5.1.2 Even rounds:

If the number of rounds $R = 2\rho$ is even, we still have $d = 2^\rho$, but as already pointed out in Section 3.1, we can take advantage of the fact that the products of the $E_{G,i}$'s have lower degree if the equations are chosen carefully. If $\ell \equiv 0 \bmod 3$, we can choose the ℓ random equations in sets of 3, such that in each set the equations are aligned under the same S-box. This reduces the algebraic degree of $\tilde{A}_G$ to $d_{\tilde{A}_G} = 4 \cdot 2^{\rho-1} \cdot \frac{\ell}{3}$ from $2^\rho \cdot \ell$. If $\ell \equiv 2 \bmod 3$, then we can again apply the above strategy of grouping the equations in sets of 3. However, we need to choose one set of cardinality 2, and we can select these which are aligned under some randomly chosen S-box. In that case, $d_{\tilde{A}_G} = 4 \cdot 2^{\rho-1} \cdot \lfloor \frac{\ell}{3} \rfloor + 3 \cdot 2^{\rho-1}$. If $\ell \equiv 1 \bmod 3$, then we have to choose one set of cardinality 1, which makes $d_{\tilde{A}_G} = 4 \cdot 2^{\rho-1} \cdot \lfloor \frac{\ell}{3} \rfloor + 2^\rho$.

Thus the degree of $\tilde{F}_G$ can be made to be around $d_{\tilde{A}_G} - n_1/3$ by judiciously choosing the random set of equations. In that case we need $J(n-n_1, d_{\tilde{A}_G} - n_1/3)$ points to evaluate $\tilde{F}_G$ on all points in B^{G_2}. Thus we need the following adjustments to the time and memory complexity:

1. We need to adjust M_{eval} to $J(n - n_1, d_{\tilde{A}_G} - n_1/3) \cdot n \cdot 2^{n_1/3}$ and $T_{eval,total}$ to $2^{n/3} \cdot J(n - n_1, d_{\tilde{A}_G} - n_1/3 - 1) \cdot T_{oracle}$.
2. Thus the adjusted memory complexity is given as:
$$MC = \log_2 N \cdot 2^{(2n-2n_1)/3} + M(n - n_1, d_{\tilde{A}_G} - n_1/3) + M_{eval} + M_{store} \tag{13}$$
3. Thus the adjusted time complexity is given as:
$$TC = T_{eval,total} + T_{int} + 2^{n/3} \cdot T_{test}. \tag{14}$$

where T_{int} is adjusted to $N \cdot 2^{n/3} \cdot T(n - n_1, d_{\tilde{A}_G} - n_1/3 - 1)$. For $n = 255, R = 6$, we have $d = 8$ after linearization. If we choose $n_1 = 9,\ \ell = 2n_1/3 + 2 = 8$, we get $d_{\tilde{A}_G} = 44$. Again choosing $N = 18$ and $\theta = 13$, we get $\epsilon_2 = 2^{-18}$. We get $TC \approx 2^{259.7}$ and $MC \approx 2^{168.57}$ bits and a success probability of $\epsilon_1 \approx 0.52$.

6 Time-Memory Tradeoffs

6.1 A note on testing solutions

In [Din21a, Appendix B], the author tests candidate solutions using a Horner-like method shown as follows. Note that the cryptosystem is described by n multivariate polynomials P_j of n variables and of some degree d. The main idea is to test together all candidates that share the same value for the t msbs $\chi_t = k_0, k_1, \dots, k_{t-1}$. Then the attacker can use of each the 2^t values of χ_t to reduce the above polynomials to just $n - t$ variables. For each value of χ_t, this takes around $\binom{n}{\downarrow d}$ bit operations per polynomial. The attacker does the same for around e such polynomials P_j to get a reduced system $\mathcal{R}$ of e polynomials in $n-t$ variables each of degree d. If Q is the original number of candidates to be tested, then on average there are $Q \cdot 2^{-t}$ candidates per distinct value of χ_t.

Since each of these $Q \cdot 2^{-t}$ candidates has the same value in the t-msbs, they can be reference by the respective $n-t$ lsbs. Evaluating one of the equations of $\mathcal{R}$ for one of these candidates would require around $\binom{n-t}{\downarrow d}$ bit-operations. Hence the complexity for all the candidates, for evaluating all e polynomials in $\mathcal{R}$ should take $Q \cdot 2^{-t} \cdot e \cdot \binom{n-t}{\downarrow d}$. For each of these candidates the attacker can evaluate each polynomial in the reduced system and eliminate it if all polynomials are not satisfied. So the total complexity to do the above for all values of the t msbs is $S_1 = 2^t \cdot \left[e \cdot \binom{n}{\downarrow d} + Q \cdot 2^{-t} \cdot e \cdot \binom{n-t}{\downarrow d}\right]$.

The number of candidates that remain after testing only e equations is $Q_1 = Q \cdot 2^{-e}$ on average. The parameters are selected so that this figure is small enough to search exhaustively. For example, the attacker can test another equation using $Q_1 \cdot \binom{n}{\downarrow d}$ operations, which brings the number of candidates down to $Q_1/2$. Thus repeating the procedure for around $\log_2 Q_1$ steps reduces the solution space to 1. So this procedure takes $S_2 = \binom{n}{\downarrow d} \cdot \left[\sum_{i=0} \frac{Q_1}{2^i}\right] \approx 2 \cdot Q_1 \cdot \binom{n}{\downarrow d}$. In [Din21a], the author shows that the complexity $S_1 + S_2$ can be ignored for 4-round LowMC, however we find that this value can no longer be ignored for 5-round LowMC onwards. For example, using the parameters in [Din21a], in 5-round LowMC with blocksize $n = 255$, the optimal value of $S_1 + S_2$ is around $2^{254.4}$ (for $e = 18,\ t = 199$).

6.2 Tradeoffs

In the rest section we will study generic time-memory trade-offs that can be applied to the algorithm to decrease the memory complexity even further. As mentioned in [Din21a, Section 4.3], a generic time-memory trade-off can be done by guessing the values of g variables and looking for roots of the equations for $n-g$ remaining variables in the subset induced by this guess. If $T(n, n_1, d)$ and $M(n, n_1, d)$ represent the time and memory complexity of the algorithm (obtained after adding the S_1+S_2 expression derived above), the complexities obtained by applying this trade-off would be $T'(n, n_1, d, g) = 2^g T(n-g, n_1, d)$ and $M'(n, n_1, d, g) = M(n-g, n_1, d)$.

In order to apply the same trade-offs for our algorithm there is only one detail that should be taken care of. As we linearize the S-boxes based on a quadratic function in the input, guessing some of the bits might induce inconsistencies with the value guessed for this quadratic function. For instance if x_0, x_1 are assigned to be 1, and the quadratic function in question is the majority function, the guess $\mathsf{maj}(x_0, x_1, x_2) = 0$ is inconsistent with the assigned values.

Circumventing this issue is quite straight forward. The trick is to guess the values of the inputs of t S-boxes from the first round ($g = 3t$ guesses), instead of guessing g variables arbitrarily. The time/memory complexity in this case would be $T'(n, n_1, d, t) = 2^{3t} T(n-3t, n_1, d)$ and $M'(n, n_1, d, t) = M(n-3t, n_1, d)$.

In order to compare our results with the ones given in [Din21a], we have computed and plotted the time and memory complexities for both the algorithms with respect to different configurations of g. We compare the complexities for $n = 129, 192, 255$ and all variants with $R \in \{2, \dots, 5\}$ rounds. We did not compare higher values of R as the complexity computed in [Din21a] in these cases is worse than gray-code assisted exhaustive search and therefore ours even with zero keybit guesses. For each of the instances, based on the value of g, we pick the value of n_1 which optimizes the time and memory complexity. The result attests a significant decrease in memory complexity, without a significant penalty in terms of time complexity. Fig. 3 demonstrates both time and memory complexities from

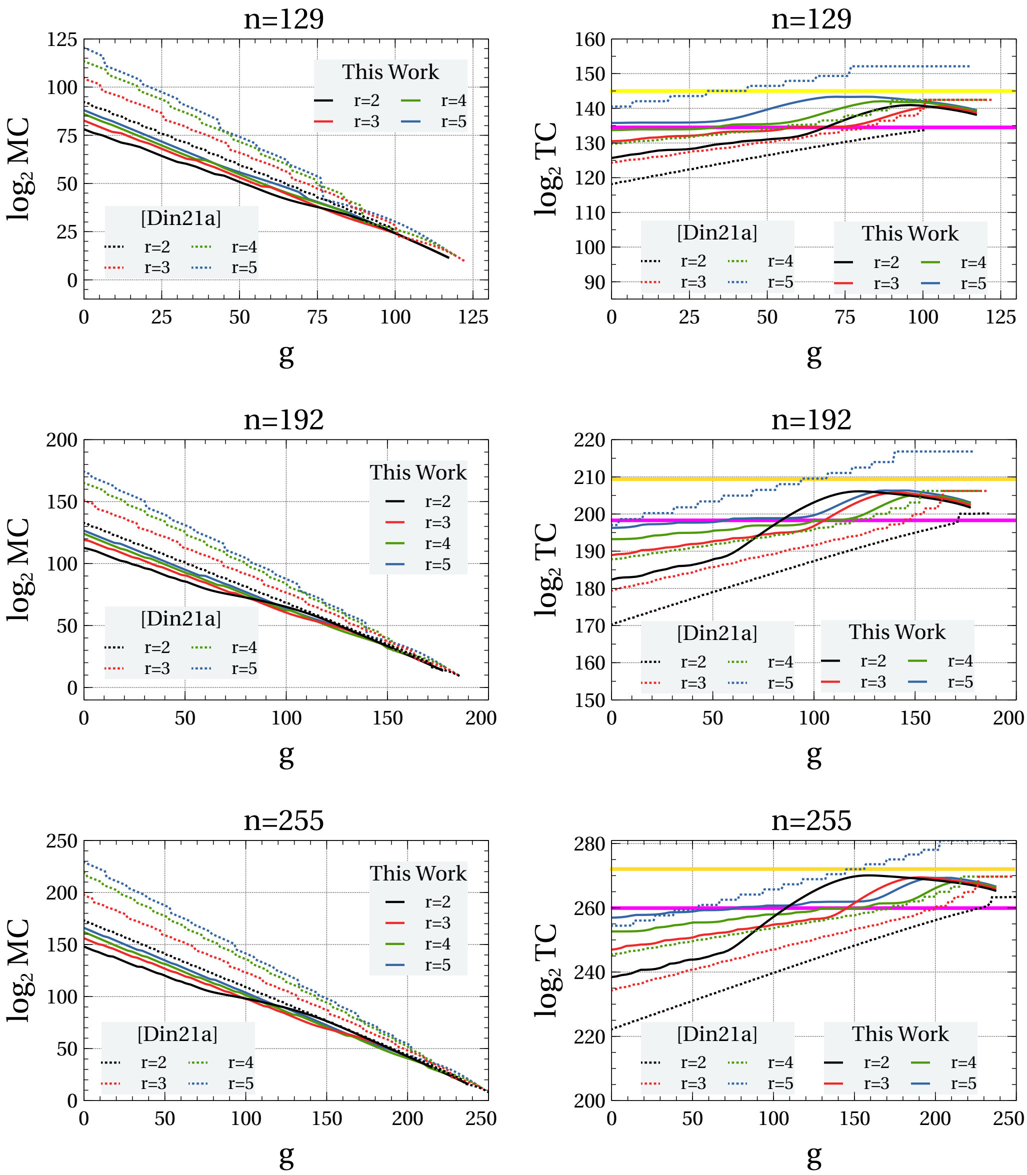


Figure 3: This figure demonstrates the time and memory complexity of the attacks presented in this work (solid lines) and [Din21a] (dotted lines) for different number of key-bits guessed (g), for different variants of the cipher with block sizes $129, 192$ and 255 and different number of rounds $R = 2, 3, 4, 5$. The figures on the left-hand side present the logarithm of the memory complexities, and the figures on the right present the logarithm of the time complexity of each attack. The thick yellow and purple lines denote the complexity of simple and gray-code accelerated exhaustive search respectively.

this work and [Din21a] side-by-side, for different number of key-bit guesses.

Upon the conclusion, we note that there are other generic time/memory tradeoffs that can be applied to reduce the memory complexity. However, as the algorithm proposed in this work and the algorithm proposed in [Din21a] use the same framework, unless a tradeoff is specifically tailored to one of the algorithms, it can be applied to the other as well.

7 Conclusion

In this paper we revisit key recovery attacks on the LowMC block cipher given a single plaintext and ciphertext pair. This attack scenario is important as directly leads to the retrieval of the signing key in the PICNIC digital signature scheme. We began the attack by linearizing the first round by guessing the value of a balanced quadratic equation in the master key bits. This tessellates the keyspace into numerous partial sets, and by limiting our key search procedure to these partial sets we can limit the memory complexity of the algorithm to just about $2^{2n/3}$ bits, while attacking some 5, 6 and 7-round instances of LowMC.

References

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 430–454, 2015. doi:10.1007/978-3-662-46800-5_17.

[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. doi:10.46586/tosc.v2020.i4.130-146.

[BBVY21] Subhadeep Banik, Khashayar Barooti, Serge Vaudenay, and Hailun Yan. New attacks on lowmc instances with a single plaintext/ciphertext pair. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 303–331. Springer, 2021. doi:10.1007/978-3-030-92062-3_11.

[BCC+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 F_2. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 203–218, 2010. doi:10.1007/978-3-642-15031-9_14.

[Din21a] Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021. doi:10.1007/978-3-030-77870-5_14.

[Din21b] Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021. doi:10.1137/1.9781611976465.151.

[DS11] Itai Dinur and Adi Shamir. An improved algebraic attack on hamsi-256. In Antoine Joux, editor, *Fast Software Encryption*, pages 88–106, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21702-9_6.

[GKRS] Lorenzo Grassi, Daniel Kales, Chistian Rechberger, and Markus Schofnegger. Survey of key-recovery attacks on lowmc in a single plaintext/ciphertext scenario. https://raw.githubusercontent.com/lowmcchallenge/lowmcchallenge-material/master/docs/survey.pdf.

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. A simple algebraic attack on 3-round lowmc. *IACR Cryptol. ePrint Arch.*, 2021:255, 2021. URL: https://eprint.iacr.org/2021/255.

[LMSI22] Fukang Liu, Willi Meier, Santanu Sarkar, and Takanori Isobe. New low-memory algebraic attacks on lowmc in the picnic setting. *IACR Trans. Symmetric Cryptol.*, 2022(3):102–122, 2022. doi:10.46586/tosc.v2022.i3.102-122.

[LPT+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 Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017. doi:10.1137/1.9781611974782.143.

[Zav] Greg Zaverucha. The picnic signaure algorithm specifications, version 3.0, available at https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf.

Appendix A: Algorithms for Efficient Möbius Transform

In this section, we present the algorithms for memory efficient Möbius Transform described in Section 4.3, and analyze its time complexity.

A.1 Algorithm 1

This algorithm interpolates the algebraic form of **F** given its evaluation on $J(n,d)$ points of its input space, where $n = 3t$ is a multiple of 3, and d is the degree of F. The first algorithm requires that the attacker be able to map **(a)** each of the $J(n,d)$ vectors $v \in \{0,1\}^{2t}$ that is into an index $j \in [0, J(n,d)-1]$ and **(b)** an operation that computes the inverse map efficiently.

We have already seen that all points included in the set of $J(n,d)$ have a special form, i.e., if $L(n,d)$ is the set of binary strings of length $t = n/3$ and hamming weight upto d. Then by a slight abuse of notation all the strings in the set $J(n,d)$ can be written as $\cup_{i=0}^{d} L(n,i) \otimes \{01,10,11\}^i$. For example if $1001 \in L(12,2)$, then $1001 \otimes [11,01]$ is defined as 11 00 00 01. Then it can be seen that the following 3^2 strings that belong to the set $J(12,2)$ are contributed by 1001:

(1) 01 00 00 01	(4) 10 00 00 01	(7) 11 00 00 01
(2) 01 00 00 10	(5) 10 00 00 10	(8) 11 00 00 10
(3) 01 00 00 11	(6) 10 00 00 11	(9) 11 00 00 11

Thus we first find a way to index all length $n/3$ strings of weight upto d. To do this we define a function **next**(u), that returns the smallest integer larger than u that has the same hamming weight as u (the authors found the subroutine at https://stackoverflow.com/questions/13542794/hamming-weight-based-indexing).

next(u)

```
lo = u & -u;                 // lowest one bit
int lz = (u + lo) & ~u;      // lowest zero bit above lo
u |= lz;                     // add lz to the set
u &= ~(lz - 1);              // reset bits below lz
u |= (lz / lo / 2) - 1;      // put back right number of bits at end
return u;
```

Define 2 arrays Ind and Ind^{-1} of $J(n,d)$ entries. We index the strings in the following manner. Note we use Ind^{-1} as a hash table where the input $2n/3$ strings u are mapped to some value in $[0, J(n,d)-1]$.

index(n,d)

```
Ind[0] = 0, Ind^{-1}[0] = 0, loc=1
for i = 1 → d
    x = 2^i -1        // smallest integer of weight i
    do
           for j = 0  → 3^d-1
               t_{d-1}, t_{d-2} ... t_0  ←  Ternary representation of j
               u = x ⊗ (1+t_{d-1}), (1+t_{d-2}), ...,  (1+t_0)
               Ind[loc] = u,
               Ind^{-1}[hash(u)] = loc
               loc = loc+1
           end for j
```

```
            x= next(x)
        while x is of less than n/3 bits
    end for i
```

Note that the runtime of the above algorithm is proportional to $J(n,d)$ and since it has to be performed only once and not for all G, this results in a small overhead which is negligible when compared to the total time complexity TC of the algorithm. We do have to store 2 additional arrays which results in a space overhead of $J(n,d)\cdot[2n/3+\log_2 J(n,d)]$ bits.

7.0.1 Möbius Transform for interpolation:

In general the in place Möbius transform has a simple and elegant structure given by $\mathbf{Möbius_1}(m)$ below, where `A[j]` is initially the evaluation of the underlying function at point `j`. After the routine is executed `A[j]` stores the co-efficient of the algebraic expression of the Boolean function indexed by the bits of `j`:

$\mathbf{Möbius_1}(m = 2n/3)$

```
for i = 0 to m-1
    e = 1 << i
    for j = 0 to 2^m-1
        if j & e != 0
           A[j]=A[j]⊕A[j⊕e]
        end if
    end for j
 end for i
```

$\mathbf{Möbius_2}(m = 2n/3)$

```
for i = 0 to m-1
    e = 1 << i
    for j = 0 to J(n,d)-1
        j' = Ind[j]
        if j' & e != 0
           j''= j'⊕e
           k = Ind^-1[hash(j'')]
           A[j]=A[j]⊕A[k]
        end if
    end for j
end for i
```

The algorithm $\mathbf{Möbius_2}(m)$ is what we propose as Algorithm 1. The algorithm is exactly the same except we account for the fact that `A[j]` now stores the evaluation of the function at point Ind[j]. If both Ind[j] and Ind[j]⊕`e` are points in $J(n,d)$ we proceed with updating the array. Note that **(a)** the `hash` computations and hence computation of `k` and, **(b)** the computation `j' & e` can be performed one time and stored in a table using $\left(\frac{2n}{3}\cdot(1+\log_2 J(n,d))\right)\cdot J(n,d) \approx \left(\frac{2n}{3}\cdot\log_2 J(n,d)\right)\cdot J(n,d)$ bits of memory (for each `j` in the set of $J(n,d)$, for each of the $2n/3$ vectors `e`, we store `k` i.e $\log_2 J(n,d)$ bits and the value of `j' & e` (1 bit)). So these computations add only a small overhead to the time complexity itself.

7.0.2 Total number of xors:

For each `e`, a location pointed to by `j'` is only overwritten if corresponding it has 1 in the location pointed to by the single one in the unit vector`e`. It can be seen that the number of such strings is $J'(n,d) = 2\cdot\sum_{i=0}^{d-1}\binom{n/3-1}{i}\cdot 3^i$. Since each such string produced by tensor multiplication with strings from $L(n/3-1,d-1)$ and inserting two of the three tuples $\{01,10,11\}$ at the location where `e` is 1. We claim that each such location pointed to by these strings are overwritten. This is true since each such string `s`, has the property that `s`⊕`e` belongs to the set of strings produced by tensoring from $L(n/3-1,d-1)$ and inserting 00, and hence these must be in $J(n,d)$. Hence the total number of xors used in the algorithm is $\frac{2n}{3}\cdot J'(n,d)$.

A.2 Algorithm 2

This algorithm will find the entire truth table of F given the vector of coefficients of its algebraic expression. The first step would be to re-index the coefficients of the algebraic expression into an array A of size $2^{2n/3}$ according a natural canonical ordering, i.e. the coefficient of the monomial $\prod x_i^{v_i}$ is written into the array index $\sum v_i \cdot 2^i$. If F is of degree d, we have already established that the associated function F is of degree $2d$. Thereafter, at the i-th step ($0 \le i < 2n/3$), the array is divided into 2^{i+1} sub-arrays and only the indices whose hamming weight is $\le 2d$ in the least significant $g = 2n/3 - i - 1$ bits in one-half of the sub-arrays are updated. Therefore the total number of xor operations for $0 \le i < 2n/3 - 2d$ is around $2^i \cdot \sum_{l=0}^{2d} \binom{2n/3-1-i}{l} = 2^i \cdot \binom{2n/3-1-i}{\downarrow 2d}$. For the remaining $2n/3 - 2d \le i < 2n/3$, the figure is exactly $2^{2n/3-2d-1}$. In [Din21a, DS11], it was shown that this sum is around $2d \cdot 2^{2n/3}$. The pseudo-code is given as follows:

Möbius₃$(m = 2n/3, deg = 2d)$

```
ρ ← 0;  b ← 1 ≪ m − 1; // all one string of length m
  for i = 0 → m − 1
        g ← m − 1 − i;  e ← 1 ≪ g;
        ρ ← ρ ⊕ e;   // cumulative sum of unit vectors
        mask ← ∼ρ & b; // mask set to 0 in i+1 MSBs i.e. 0^{i+1}1^{m−i−1}
        for j ← 0 → 2^g − 1
              for k ← 0 → 2^i − 1
                    ind ← j + k ∗ (2^{g+1});
                    if hw(ind & mask) ≤ deg
                         A[ind ⊕ e] ← A[ind ⊕ e] ⊕ A[ind];
                    end if
              end for k
        end for j
   end for i
```