IACR Transactions
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–23. DOI:XXXXXXXX

LowMC Cryptanalysis Challenge: 2nd Round

Subhadeep Banik, Khashayar Barooti, Serge Vaudenay

LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{subhadeep.banik,khashayar.barooti,serge.vaudenay}@epfl.ch

Abstract. Arguably one of the main applications of the LowMC family ciphers is in the post-quantum signature scheme PICNIC. Although LowMC family ciphers have been studied from a cryptanalytic point of view before, none of these studies were directly concerned with the actual use case of this cipher in PICNIC signature scheme. Due to the design paradigm of PICNIC, an adversary trying to perform a forgery attack on the signature scheme instantiated with LowMC would have access to only a single given plaintext/ciphertext pair, i.e. an adversary would only be able to perform attacks with data complexity 1 in a known-plaintext attack scenario. This restriction makes it impossible to employ classical cryptanalysis methodologies such as differential and linear cryptanalysis. In this paper we introduce a set of key-recovery attacks, both in known-plaintext model and of data complexity 1 for two variants of LowMC, both instances of the LowMC cryptanalysis challenge.

Keywords: No keywords given.

1 Introduction

In recent years, a significant amount of attention has been drawn towards designing post-quantum cryptographic primitives, such as digital signature schemes. There have been several design ideas for quantum-secure signature schemes, some of which are based on lattice problems, supersingular isogenies or schemes providing information-theoretic security.

PICNIC [CDG+17] is a highly tweakable signature scheme based on an MPC-in-head paradigm, currently in the third round of NIST post-quantum cryptography competition [nis]. The authors propose several different parameters for various security levels and applications. Instantiating PICNIC requires a hard to invert function which has low computational overhead when computed in a multi-party manner. This overhead relies heavily on the number of non-linear operations needed to compute the function, i.e. number of multiplications.

LowMC [ARS+15] is an efficient block-cipher tailored specifically for FHE and MPC usage, aiming to minimize the number of multiplications. LowMC uses a quadratic S-box operating on 3 bit inputs, and for each output bit of the S-box, a single multiplication is needed. The S-box is then followed by an affine layer, operating on the whole block, followed by a round-key addition.

The low multiplication count makes LowMC a fairly suitable choice for PICNIC instantiation. Both the LowMC instance and the MPC steps of PICNIC have multiple parameters to set for different use cases, but this also makes it challenging to select the optimal parameters, one of which is the number of rounds/S-boxes. As mentioned before, the computational overhead of performing a LowMC encryption in a multi-party manner is heavily dependent on the number of multiplications needed. Moreover, reducing the number of multiplications too much will lead to security issues, which makes it challenging to find a good parametrization to maximize efficiency without harming the security.

In ICISC 2015 Dobraunig et al. [DEM15] proposed an attack on LowMC family of block ciphers, based on cube attack strategies. The authors proposed an algorithm which successfully recovers the key of the round reduced version of the cipher, aiming for 80-bit security. Later in FSE 2018, Rechberger et al. [RST18] proposed a meet-in-the-middle style attack, based on possible output differentials, given an input differential, which affects the security of the variants of LowMCv2 with partial S-box layers drastically. In [LIM20] some results on LowMC were reported building on the techniques of [RST18], albeit with higher data complexities, which naturally do not apply to the PICNIC scenario. For a survey of key recovery attacks on LowMC, readers may check the survey done by Rechberger et al. [DKRS]. As mentioned, one of the main use cases of LowMC, is the PICNIC post quantum signature scheme. Due to PICNIC's algebraic composition, the scheme would be trivially forged by a key recovery attack on LowMC that uses only a single pair of plaintext/ciphertext. In other words only attacks with data complexity one directly affect the security of the signature scheme.

In May 2020, Rechberger et al. started a cryptanalysis challenge for LowMC key-recovery, specifically for the PICNIC use case, meaning the attacks should be performed using a single ciphertext/plaintext pair in a known-plaintext model. In this paper we propose two attacks with data complexity one for two parameter sets of the LowMC challenge. Our attacks successfully break the two-round version of LowMC with full S-box layer, and the partial S-box variant with $0.8 \times \lfloor n/s \rfloor$ rounds, where n denotes the block size, and s denotes the number of S-boxes used in each round.

We continue by giving a brief high-level description of the PICNIC signature, and intuitively demonstrate why a data-complexity one key-recovery attack on LowMC cipher would lead to a PICNIC signature forgery.

1.1 PICNIC Signature Scheme

PICNIC signature is built using Fiat-Shamir transformation of a sigma protocol based on the MPC-in-head paradigm by Ishai et al. [IKOS07]. The high-level idea is as follows, imagine we have a multi-party computation of a function f. Each player has a share of the input x, and the output $y = f(x)$ is publicly known. The prover simulates all players and commits to all the states and transcripts. Later the verifier is allowed to corrupt a random subset of players, having access to their full state. Having this information in hand, the verifier can check whether the computation was done correctly from the corrupted players' perspective.

In the case of PICNIC this paradigm is instantiated using LowMC block cipher. Let $E(K, pt)$ be the LowMC encryption of the plaintext pt using the key K. The function f in the previous paradigm is instantiated as $E(*, pt)$ for a public plaintext pt. The plaintext/ciphertext pair (pt, ct) is used as the public key of the signature scheme (verification key) and encryption key K is used as the secret key (signing key). If an adversary can recover the encryption key given only a single ciphertext, plaintext pair (ct, pt) i.e. the public key of the signature scheme, then in effect he computes the secret signing key. This allows him to forge a signature by following exactly the honest prover protocol with the recovered signing key. This demonstrates that a data complexity one key recovery attack on LowMC block cipher leads to a signature forgery on PICNIC.

The size of the signature and the efficiency of the PICNIC signing algorithm heavily rely on the number of the multiplications (and gates) used in the encryption circuit. This has driven some interest towards finding an optimal number of and gates, to keep the desired security level and to provide the best level of efficiency, and recently Rechberger et al. announced a cryptanalysis challenge for LowMC family of block ciphers, specifically for the PICNIC use case, i.e. only one plaintext ciphertext provided to the attacker.

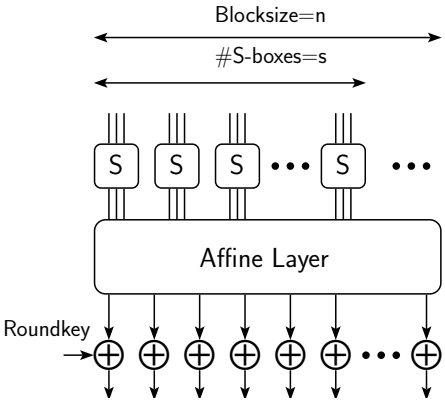

Figure 1: LowMC Round Function

1.2 Contribution and Organization of the Paper

In this paper we introduce two cryptanalysis results for two instances of the LowMC cryptanalysis challenge, namely the 2 round LowMC with full S-box layer and the partial S-box layer with $0.8 \times \lfloor n/s \rfloor$ number of rounds. Both attacks are performed in a known plaintext/ciphertext attack model (KPA/KCA), and both attacks have data complexity 1. These restrictions make the task fairly hard, as the classical cryptanalysis methods such as linear and differential cryptanalysis can not be employed. In this paper, we first show how to efficiently linearize the LowMC S-box by guessing only one quadratic expression in the S-box inputs. Leveraging this fact we describe a series of attacks on some instances of LowMC given in the cryptanalysis challenge. Our results are summarized in Table 1.

In section 2, we describe the algebraic form of the LowMC round function and furnish some details of the LowMC cryptanalysis challenge. Section 3 demonstrates an efficient method to linearize the LowMC S-box which will be used later. Section 4 introduces a key-recovery attack on 2-round full S-box layer and partial S-box with $0.8 \times \lfloor n/s \rfloor$ round variants of LowMC. In Section 5, we introduce a meet in the middle attack on the same variants of LowMC, and in Section 6 we demonstrate how this approach can be optimized using the 3-xor problem by a factor of $\sqrt{n/2}$. In Sections 7 and 8, we describe improved 2-stage MITM attacks on 2-round full S-box and $0.8\lfloor \frac{n}{s} \rfloor$-round partial S-box layers respectively. We conclude the paper in Section 9.

2 The LowMC challenge

The LowMC round function is a typical SPN construction given in Figure 1. It consists of an n-bit block undergoing a partial substitution layer consisting of s 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 the roundkey which is again the product of the n-bit master secret key K with an $n \times n$ invertible matrix. As in most SPN constructions, a 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.

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

- $n = 128, \quad s = 1$

Table 1: Summary of results. Note for the complexity is given in "evaluations of a quadratic expression" * As explained in Section 4, these are best case complexities that occur for around 29% of the LowMC instances.

Instance	n	s	r	Type of Attack	Complexity	Section
Full S-box layer	129	43	2	Linearization + MITM	2^{109}	5
	192	64			2^{161}	
	255	85			2^{214}	
Partial Sbox layer	128	1	$0.8 \times \lfloor \frac{n}{s} \rfloor$	Linearization+ MITM	2^{124}	5*
	192	1			2^{186}	
	256	1			2^{248}	
Partial Sbox layer	128	10	$0.8 \times \lfloor \frac{n}{s} \rfloor$	Linearization + MITM	2^{124}	5*
	192	10			2^{186}	
	256	10			2^{248}	
Full S-box layer	129	43	2	Linearization + MITM+3-xor	2^{106}	6
	192	64			2^{158}	
	255	85			2^{211}	
Full S-box layer	129	43	2	2 stage MITM	2^{88}	7
	192	64			2^{130}	
	255	85			2^{172}	
Partial Sbox layer	128	1	$0.8 \times \lfloor \frac{n}{s} \rfloor$	2 stage MITM	2^{104}	8
	192	1			2^{156}	
	256	1			2^{207}	
Partial Sbox layer	128	10	$0.8 \times \lfloor \frac{n}{s} \rfloor$	2 stage MITM	2^{98}	8
	192	10			2^{154}	
	256	10			2^{202}	

- $n = 128, \quad s = 10$
- $n = 129, \quad s = 43$ (full S-box layer)
- $n = 192, \quad s = 1$
- $n = 192, \quad s = 10$
- $n = 192, \quad s = 64$ (full S-box layer)
- $n = 256, \quad s = 1$
- $n = 256, \quad s = 10$
- $n = 255, \quad s = 85$ (full S-box layer)

The number of rounds r for instances with the full 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 $1.2 \times \lfloor \frac{n}{s} \rfloor$. The key length k for all instances is n bits. 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 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 PT/CT pair. As such we use the terms master key/whitening key interchangeably.

3 Preliminaries

The algebraic forms of the 3 output bits of the S-box S used in LowMC are given by the following expressions:

$$\begin{aligned} s_0 &= x_0 + x_1 \cdot x_2, \\ s_1 &= x_0 + x_1 + x_0 \cdot x_2, \\ s_2 &= x_0 + x_1 + x_2 + x_0 \cdot x_1 \end{aligned}$$

Similarly the inverse S-box S^{-1} is given by the following expressions

$$\begin{aligned} t_0 &= x_0 + x_1 + x_1 \cdot x_2, \\ t_1 &= x_1 + x_0 \cdot x_2, \\ t_2 &= x_0 + x_1 + x_2 + x_0 \cdot x_1 \end{aligned}$$

Let us for example take f to be the majority function computed on the inputs of the 3 input bits, i.e. $f = x_0 \cdot x_1 + x_1 \cdot x_2 + x_0 \cdot x_2$. Then the expressions of the S-box can be rewritten as

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

This means that if we guess the value of the single expression f (0 or 1), then the entire S-box becomes an affine function in the input bits. The same holds for the inverse S-box. In fact we can replace f with any balanced 3-variable Boolean function of degree 2, and still get the same results as we prove in the following lemma.

Lemma 1. *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.*

Proof. The general expression for a 3 variable quadratic Boolean function is

$$f = A + Bx_0 + Cx_1 + Dx_2 + Ex_0 \cdot x_1 + Fx_1 \cdot x_2 + Gx_0 \cdot x_2.$$

The only non-linear terms in the expression of the LowMC S-box are $x_0 \cdot x_1,\ x_1 \cdot x_2,\ x_0 \cdot x_2$. Thus if there exists a Boolean function of the above form, which when multiplied with different linear functions can produce each of the terms $x_0 \cdot x_1,\ x_1 \cdot x_2, x_0 \cdot x_2$, then we are done. Thus the necessary and sufficient conditions required to achieve the above is to prove the existence of 3 affine Boolean functions $g_i = a_i x_0 + b_i x_1 + c_i x_2 + d_i,\ \forall i \in [0, 2]$, such that

$$\begin{aligned} f \cdot g_0 &= x_0 \cdot x_1 + l_0(x_0, x_1, x_2) \\ f \cdot g_1 &= x_1 \cdot x_2 + l_1(x_0, x_1, x_2) \\ f \cdot g_2 &= x_0 \cdot x_2 + l_2(x_0, x_1, x_2) \end{aligned}$$

where l_0, l_1, l_2 are some affine functions on x_0, x_1, x_2. If these functions g_i exist, we can write each of the three output bits of the LowMC S-box as

$$x_0 + f \cdot g_1 + l_1,\ x_0 + x_1 + f \cdot g_2 + l_2,\ x_0 + x_1 + x_2 + f \cdot g_0 + l_0$$

So in order for the first equation to be satisfied, we need that the product of f and g_0 produces coefficients $0, 1, 0, 0$ for the terms $x_0 \cdot x_1 \cdot x_2,\ x_0 \cdot x_1,\ x_1 \cdot x_2,\ x_0 \cdot x_2$ respectively. In matrix form this can be written as $\mathbf{M} \cdot [a_0,\ b_0,\ c_0,\ d_0]^T = [0, 1, 0, 0]^T$, where

$$\mathbf{M} = \begin{bmatrix} F & G & E & 0 \\ C+E & B+E & 0 & E \\ 0 & D+F & C+F & F \\ D+G & 0 & B+G & G \end{bmatrix}$$

Similarly the other 2 equations can be written as $\mathbf{M} \cdot [a_1,\ b_1,\ c_1,\ d_1]^T = [0, 0, 1, 0]^T$ and $\mathbf{M} \cdot [a_2,\ b_2,\ c_2,\ d_2]^T = [0, 0, 0, 1]^T$. It is therefore clear that for the equations to have a solution we need $\mathbf{M}$ to be invertible. Since the number of 3-variable quadratic Boolean functions f is just 2^7, we can perform the following small computer exercise: we can construct the matrix $\mathbf{M}$ for each function f and test whether it is invertible or not. We found that all functions f for which $\mathbf{M}$ is invertible, are exactly the functions that are balanced.

□

For example, if we take $f = s_0 = x_0 + x_1 \cdot x_2$, the S-box functions can be written as

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

4 Cryptanalysis by Linearization

The first technique to break LowMC by linearization is for instances for which the total number of S-boxes is less than the key length. This occurs for the following cases

1. All instances of full S-box layer with number of rounds = 2.
2. All instances of partial S-box layer with number of rounds = $0.8 \times \lfloor \frac{n}{s} \rfloor$.

The idea is as follows. We guess the value of the majority function at the input of all the S-boxes in the encryption circuit. When we do so the expression relating the plaintext and ciphertext becomes a linear expression in the key variables, i.e. of the form

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

where A is an $n \times n$ matrix over $GF(2)$. Thus the key can be found using Gaussian elimination. After this a wrong key can be discarded by simply recalculating the encryption function with the derived key and plaintext and checking if the result equals the given ciphertext or not. Of course, we need not compute the full encryption: a key can be discarded as soon as the majority function computed at the input of one of the s-boxes differs from the value used to linearize the circuit. If the total number of s-boxes in the circuit is t, then the worst case complexity of the process is 2^t gaussian eliminations calculations. For example this is 2^{86} for the LowMC instance with $n = 129, s = 43, r = 2$. However note that there is an added cost in this process. For any guess of the majority values, the matrix A computed above may not necessarily be invertible. If the dimension of the kernel of the matrix A is d_A, then we can see that $O(2^{d_A})$ keys would satisfy any equation of the form $A \cdot K = const$. Thus the verification would require running the verification for 2^{d_A} candidate keys. Moreover, we did not find any easy way to find a closed form for any bound on d_A.

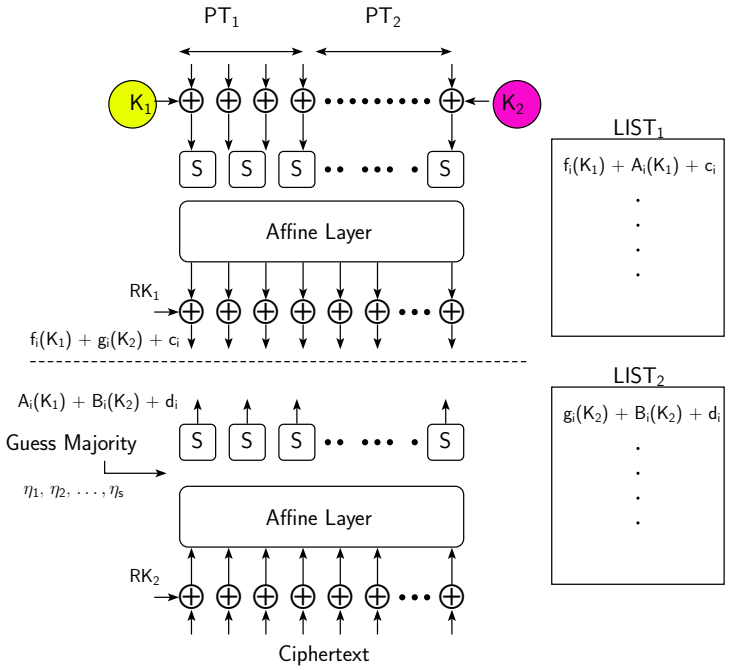

Figure 2: Meet in the Middle

5 Meet in the Middle approach

5.1 2 round full S-box layer

The complexity of the attack in the previous section was measured in terms of number of Gaussian eliminations. Even while bypassing the Gaussian Elimination method, the algorithm will still require an additional computational step (evaluating all elements in the kernel of A) and there is no easy way of finding a closed form of its value/lower bound. In this section we present attacks whose complexity is measured in much simpler and more tangible metric: "number of evaluations of a quadratic expression in keybits". We describe a meet in the middle approach for the two-round variant of LowMC. The idea is to first split the key into two parts $K_1 = [k_0, \ldots, k_{t-1}]^T$ and $K_2 = [k_t, \ldots, k_{n-1}]^T$, each of around $t \approx \frac{n}{2}$ bits. By guessing the majority bits (or any other balanced quadratic function) of the second layer S-box we can make the second round linear as described above. After this it is possible to adopt a meet in the middle approach, by guessing first the K_1 value and making a list based on each guess. We later independently guess K_2 and creating a list based on the guessed values and search for a collision in the obtained lists.

The idea is as follows. As proven in Lemma 1, if we know the value of a balanced quadratic boolean function in the input bits of each Sbox, i.e. the majority, we can write the S-box as an affine function in the input bits. The same argument holds for the inverse S-box (since the inverse S-box is also a quadratic permutation over $\{0,1\}^3$). Again let us denote by R_1, R_2 the first and second round functions i.e. $R_1(pt + RK_0, RK_1) = x$ and $R_2(x, RK_2) = ct$, where x denotes the n-bit input to the second round and RK_1, RK_2 denotes the first, second round keys, respectively, which are of course linear functions of the original key $K = RK_0$. As shown in Figure 2, we start with the ciphertext backwards and try to reach the state at the input to the second round. To do this we first perform the inverse affine function operation on the vector $ct \oplus RK_2$ (where RK_2 is expressed in terms of K_1 and K_2). Thereafter we guess the s majority bits $\eta_1, \ldots, \eta_s$ at the input of the second round inverse S-boxes to linearize R_2. After this, each bit of x can be written

as an affine function of the key and the ciphertext. In fact denoting each bit of x as x_i, we can further write $x_i = A_i(K_1) + B_i(K_2) + d_i, \forall\ i \in [0, n-1]$, where each $A_i,\ B_i$ are linear functions over K_1, K_2 and d_i is a single bit constant.

Similarly it is possible to compute x from the plaintext in the forward direction. Even if we do not guess the majority of the first round s-boxes, K_1 and K_2 can be chosen such that the bits of K_1 and K_2 are never multiplied in the first round function. For example for $n = 129$, K_1 can be taken to be the first $t = 3 \times \lfloor s/2 \rfloor = 63$ bits of the key and K_2 to be the remaining 66 bits. The only source of non-linearity in the first round are the S-boxes, and each S-box either gets the bits of K_1 or K_2 as inputs and so K_1 and K_2 are not mixed in a multiplicative sense in this round. This being the case, after the affine layer and addition of RK_1, each bit x_i can be written as $f_i(K_1) + g_i(K_2) + c_i$ where each f_i, g_i are at most quadratic functions over K_1, K_2 and c_i is a single bit constant. Given the equality $x_i = f_i(K_1) + g_i(K_2) + c_i = A_i(K_1) + B_i(K_2) + d_i$, we can rearrange the terms to get

$$f_i(K_1) + A_i(K_1) + c_i = g_i(K_2) + B_i(K_2) + d_i,\ \forall\ i \in [0, n-1]$$

We are now ready to state the attack. Let the plaintext be $pt = [pt_0, pt_1 \ldots, pt_{n-1}]$, and $ct = [c_0, c_1, \ldots, c_{n-1}]$ be the corresponding ciphertext. Take $t = 3 \times \lfloor s/2 \rfloor \approx \frac{n}{2}$. We proceed as follows:

1. Calculate the functional forms of f_i, g_i and c_i for all $i \in [0, n-1]$.
2. Guess the values $\eta_1, \ldots, \eta_s$. This step is done 2^s times in the worst case.
   - Compute A_i, B_i, d_i for all $i \in [0, n-1]$ using the guessed values.
   - For all possible values of K_1, create a hash table LIST_1 indexed by the n-bit vector $[f_i(K_1) \oplus A_i(K_1) \oplus c_i],\ \forall\ i \in [0, n-1]$. We need 2^t operations in this step.
   - For all possible values of K_2, create a hash table LIST_2 indexed by the n-bit vector $[g_i(K_2) \oplus B_i(K_2) \oplus d_i],\ \forall\ i \in [0, n-1]$. We need 2^{n-t} operations in this step.
   - Find a collision between LIST_1 and LIST_2.
   - When a collision is found for K_1 and K_2 check if the majority bits are consistent with the guess of the key. If yes, this key is in fact the encryption key. Otherwise try another guess of $\eta_1, \ldots, \eta_s$.

In practice, 2 hash tables are not necessary. The attacker can insert each new vector of LIST_1 and LIST_2 into a single hash table and wait until a collision between elements of LIST_1 and LIST_2 is found. For each set of majority guesses, the complexity of the attack is dominated by finding a collision between two lists of length 2^t and 2^{n-t} each. So for $n = 129$, we can take $t = 63$ (key bits added before the first 21 S-boxes) and $n - t = 66$. The total complexity of the attack is $O\left(2^s \times (2^t + 2^{n-t})\right)$, which for the $n = 129$ bit version is around $2^{43+66} = 2^{109}$.

5.2 MITM on partial S-box layers

In order to perform a MITM on the partial S-box layer instances of LowMC, we rearrange the first r_1 and final r_3 rounds so that the total number of different key bits involved in these rounds is $3s$ per round. The transformations are shown in Figures 3, 4 and are similar to the ones used in [RST18]. In fact the transform used in the backward direction (see Fig 4) is exactly same as the one used in [RST18, Fig.1]. The idea is that the affine layer and key addition are interchangeable. Since if L is a linear function, we

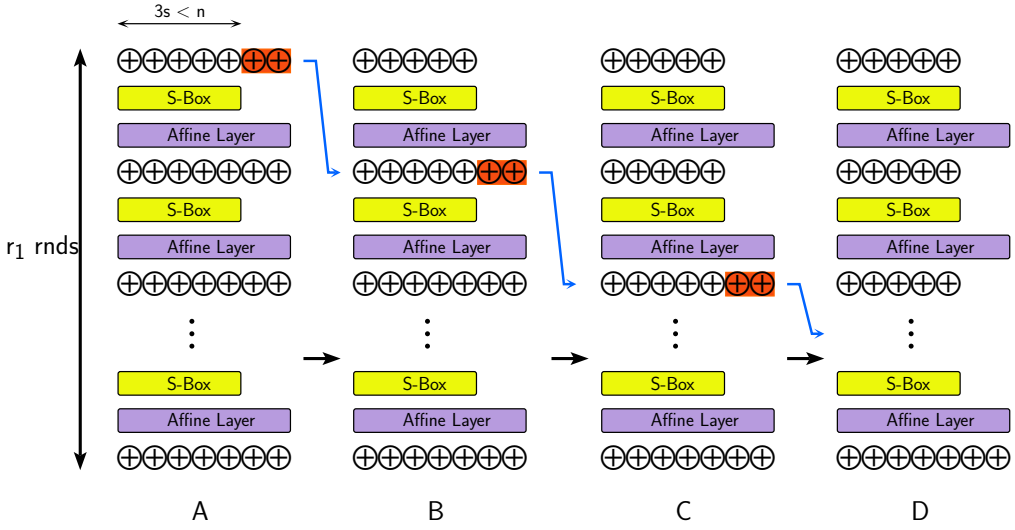

Figure 3: Transforming the round function in the first r_1 rounds. From A $\rightarrow$ B, the key material not added to bits input to the S-box in round 1 (shown in orange background) are carried to the next round, through the affine layer and merged with the round key in round 2. B $\rightarrow$ C $\rightarrow$ D do the same from the second round onwards.

have $L(x) + K = L(x + L^{-1}(K))$ and similarly $L(x + K) = L(x) + L(K)$. Hence the key addition can be moved before or after the affine layer as required, by multiplying the round key by the appropriate matrix. Fig 3 further shows how to transform the first r_1 rounds.

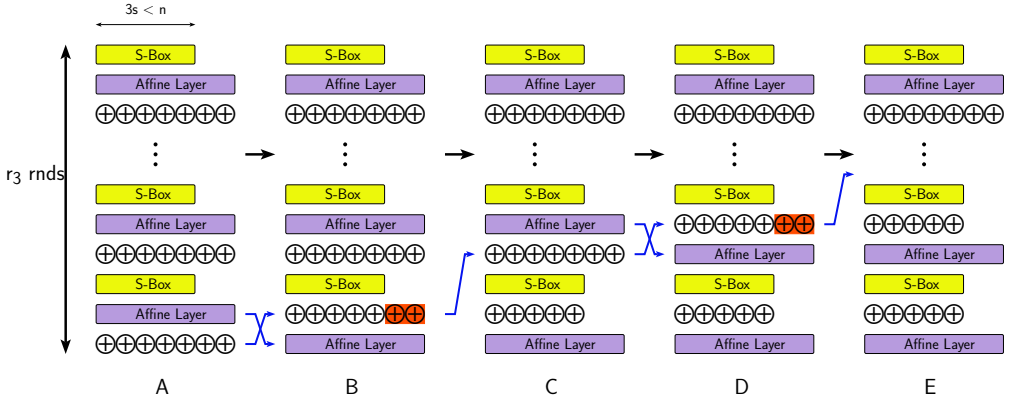

Figure 4: Transforming the round function in the final r_3 rounds. A $\rightarrow$ B flips the order of the last round Affine layer and round key xor. B $\rightarrow$ C takes the bits of the last round key that are not added to S-box outputs (shown in orange background), and brings them back by 1 round and merges it with the penultimate round key. C $\rightarrow$ D flips the order of the Affine layer and round key of the penultimate round, and D $\rightarrow$ E generalizes the process from this point onwards.

We partition the $r = r_1 + r_2 + r_3$ rounds of LowMC into the first r_1, middle r_2 and final r_3 rounds, and further transform the first r_1 and the final r_3 rounds so that each round has only $3s$ keybits. If $r_1 = r_3 = \lfloor \frac{n}{6s} \rfloor$, then there are a total of n keybits in these rounds. Naming these keybits as $\kappa_0, \kappa_1, \ldots, \kappa_{n-1}$. Let us assume that these n keybits result from linearly independent expressions on the master key bits (in the next subsection we will see what happens when this is not the case). Then it is not difficult to

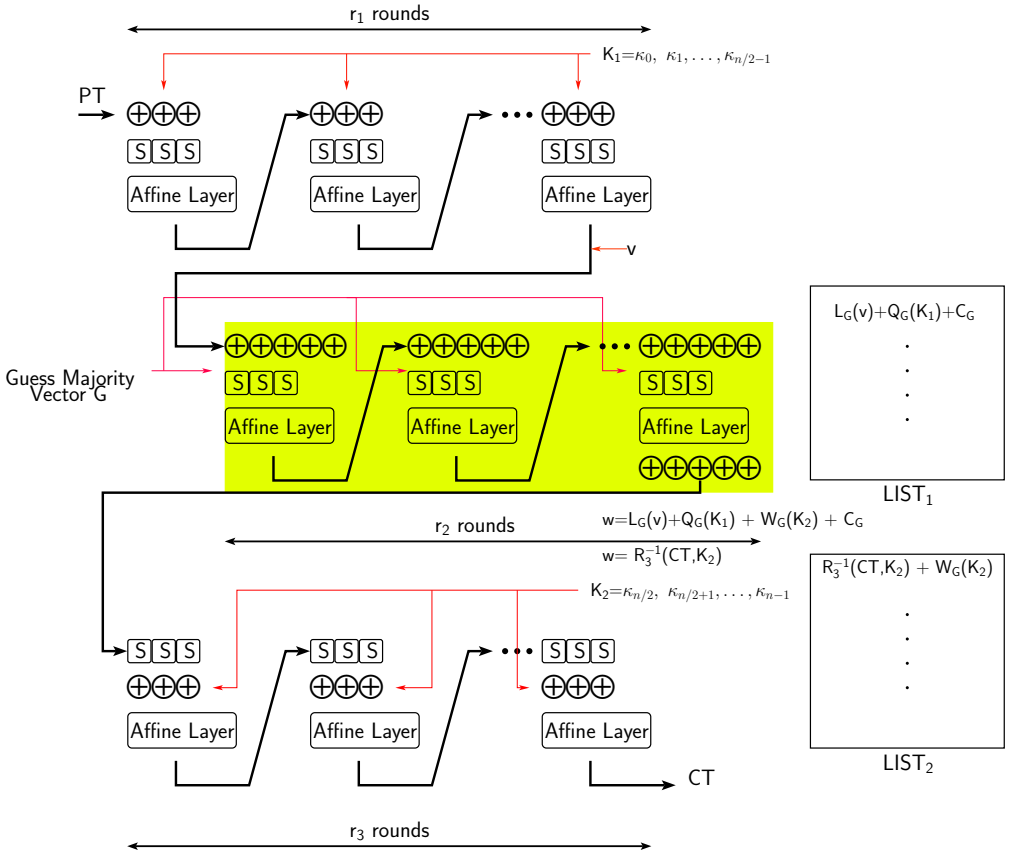

Figure 5: MITM on $r_1 + r_2 + r_3$ rounds with partial S-box layers. Note that the first r_1 and last r_3 rounds have been transformed as per the procedures explained in Figures 3, 4. We guess the majority bits at the S-box inputs in the middle r_2 rounds (shown against yellow background) so that they become affine.

see that all the keybits in the middle $r_2 = 0.8\lfloor \frac{n}{s} \rfloor - \lfloor \frac{n}{3s} \rfloor$ rounds can be written as linear functions of $\kappa_0, \kappa_1, \ldots, \kappa_{n-1}$. Now let us divide the keybits into $K_1 = [\kappa_0, \kappa_1, \ldots, \kappa_{n/2-1}]$ and $K_2 = [\kappa_{n/2}, \kappa_{n/2+1}, \ldots, \kappa_{n-1}]$ where K_1 and K_2 are the keybits used in the first r_1 and the final r_3 rounds respectively.

Now if all the $s \cdot r_2$ majority bits of the middle r_2 rounds are guessed, then the transformation in the middle r_2 rounds becomes completely affine. If G is the vector of these $s \cdot r_2$ majority bits, let us denote this affine transformation in the middle rounds as $L_G(x) + Q_G(K_1) + W_G(K_2) + C_G$, where L_G is a linear function from $\{0,1\}^n \rightarrow \{0,1\}^n$ and Q_G, W_G are linear functions over $\{0,1\}^{n/2} \rightarrow \{0,1\}^n$ and C_G is an n-bit constant. Let v be the n-bit vector obtained by executing the r_1 forward rounds by guessing some value of K_1, and let w be the vector obtained after $r_1 + r_2$ rounds. Then after guessing G we have $w = L_G(v) + Q_G(K_1) + W_G(K_2) + C_G$. Now w can also be obtained by guessing K_2 and executing the inverse of the final r_3 rounds on the ciphertext. If R_3 denotes the transformation in the last r_3 rounds, we have $w = R_3^{-1}(ct, K_2)$ So we have $R_3^{-1}(ct, K_2) = L_G(v) + Q_G(K_1) + W_G(K_2) + C_G$. Rearranging terms we have

$$R_3^{-1}(ct, K_2) + W_G(K_2) = L_G(v) + Q_G(K_1) + C_G$$

Then our meet in the middle algorithm will proceed as follows.

1. Guess the vector G of the $s \cdot r_2$ majority values in the middle rounds. Find the

functions L_G, W_G, K_G and C_G. This step is done $2^{s \cdot r_2}$ times in the worst case.

- For all possible values of K_2, create a hash table LIST_2 indexed by the n-bit vector $R_3^{-1}(ct, K_2) + W_G(K_2)$. We need $2^{n/2}$ operations in this step.
- For all possible values of K_1, create a hash table LIST_1 indexed by the n-bit vector $L_G(v) + Q_G(K_1) + C_G$. We need $2^{n/2}$ operations in this step.
- Find a collision between LIST_1 and LIST_2.
- When a collision is found for K_1 and K_2 check if the majority bits are consistent with the guess of the key. If yes, this key is in fact the encryption key. Otherwise try another guess of G.

The procedure has been explained diagrammatically in Figure 5. Again as explained before, 2 hash tables are not necessary in practice. The attacker can insert each new vector of LIST_1 and LIST_2 in a single hash table and wait till a collision between elements of LIST_1 and LIST_2. The majority of the computational complexity is taken by the guessing of G and computing $R_3^{-1}(ct, K_2) + W_G(K_2)$ for each guess of K_2 and $L_G(v) + Q_G(K_1) + C_G$ for each guess of K_1. This part takes $2^{s \cdot r_2} \cdot 2^{1+n/2} \approx 2^{s \cdot r - n/3 + n/2} = 2^{rs+n/6}$. For $r = 0.8\lfloor \frac{n}{s} \rfloor$, this complexity is around $2^{29n/30}$.

5.3 When all the key expressions $\kappa_i,\ i \in [0, n-1]$ are not linearly independent

Note that each κ_i is a linear expression in the n master key bits, and so it may turn out that the n linear expressions for $\kappa_i,\ i \in [0, n-1]$ are not linearly independent. Assuming each κ_i is a random linear expression, the probability that they are linearly independent is the same as the probability that a random $n \times n$ matrix over $GF(2)$ is invertible. In fact it is a well known result in discrete mathematics, that this probability is around 0.29 as n becomes large.

When all the κ_i's are not linearly independent, then we can not write the round keys in the middle r_2 rounds as linear expressions of the κ_i's. And if this happens, then naturally the attack as outlined in the previous subsection can not be applied. In that case how do you proceed with the attack?

1. Let us assume that for some $r_1,\ r_3$, the total rank of the $3 \cdot s \cdot (r_1 + r_3) \times n$ matrix containing the linear expressions (in terms of the master key) for all the keybits κ_i used in these rounds be equal to λ. We have already seen that that when $3 \cdot s \cdot (r_1 + r_3) = n$, the probability that $\lambda = n$ is 0.29. The probability that $\lambda = t$ is given by the expression $2^{-n \cdot t} \prod_{i=0}^{t-1} \left(1 - \frac{2^i}{2^n}\right)$. Therefore the probability that $t \geq n-1,\ n-2,\ n-3$ is around 0.58, 0.77, 0.88 respectively (for large enough n).

2. In such an event the attacker should choose suitable values of $r_1,\ r_3$ such that the value of $\lambda = 3 \cdot s \cdot (r_1 + r_3)$.

3. Let $K = [\kappa_0, \kappa_1, \ldots, \kappa_{\lambda-1}]$ be the corresponding keybits whose linear expressions are linearly independent. Let K_1 be the subset of these keybits used in the first r_1 rounds, K_2 be the subset of these keybits used in the last r_3 rounds. Choose $K_3 = [\kappa_\lambda, \kappa_{\lambda+1}, \ldots, \kappa_{n-1}]$ as random linear expressions of the master key such that the expressions for $K_1,\ K_2,\ K_3$ are linearly independent. After this step all round keybits can be written as linear expressions in $K_1,\ K_2,\ K_3$.

4. After guessing G, the vector of the middle $s \cdot r_2$ majority bits, the middle r_2 rounds become completely affine. Again if v is the vector that is the output of the first r_1 rounds, the output vector w of the first $r_1 + r_2$ rounds can be written

as $w = L_G(v) + Q_G(K_1) + W_G(K_2) + E_G(K_3) + C_G$, where L_G, Q_G, E_G are linear functions and C_G is an n-bit constant.

5. Since w can be computed from the ciphertext backwards as $w = R_3^{-1}(ct, K_2)$ So we have $R_3^{-1}(ct, K_2) = L_G(v) + Q_G(K_1) + W_G(K_2) + E_G(K_3) + C_G$. Rearranging terms we have $R_3^{-1}(ct, K_2) + W_G(K_2) = L_G(v) + Q_G(K_1) + E_G(K_3) + C_G$. Let us partition K_3 into two disjoint sets K_{31} and K_{32} so that the number of bits in $K_1 \cup K_{31}$ and $K_2 \cup K_{32}$ are almost same. We write $E_G(K_3) = E_G^1(K_{31}) + E_G^2(K_{32})$. Rearranging terms further we have

$$R_3^{-1}(ct, K_2) + W_G(K_2) + E_G^2(K_{32}) = L_G(v) + Q_G(K_1) + E_G^1(K_{31}) + C_G$$

After this our meet in the middle algorithm will proceed as follows.

1. Choose suitable values of $r_1,\ r_3$ such that the value of $3 \cdot s \cdot (r_1 + r_3) = \lambda$.
2. Choose $K_3 = [\kappa_\lambda, \kappa_{\lambda+1}, \ldots, \kappa_{n-1}]$ as random linear expressions of the master key such that the expressions for $K_1,\ K_2,\ K_3$ are linearly independent.
3. Partition K_3 into two disjoint sets K_{31} and K_{32} so that the number of bits in $K_1 \cup K_{31}$ and $K_2 \cup K_{32}$ are almost same.
4. Guess the vector G of the $s \cdot r_2$ majority values in the middle rounds. Find the functions $L_G, W_G, K_G, E_G^1, E_G^2$ and C_G. This step is done $2^{s \cdot r_2}$ times in the worst case.
   - For all possible values of $K_2 \cup K_{32}$, create a hash table LIST_2 indexed by the n-bit vector $R_3^{-1}(ct, K_2) + W_G(K_2) + E_G^2(K_{32})$. We need around $2^{n/2}$ operations in this step.
   - For all possible values of $K_1 \cup K_{31}$, create a hash table LIST_1 indexed by the n-bit vector $L_G(v) + Q_G(K_1) + E_G^1(K_{31}) + C_G$. We need around $2^{n/2}$ operations in this step.
   - Find a collision between LIST_1 and LIST_2.
   - When a collision is found check if the majority bits are consistent with the guess of the key. If yes, this key is in fact the encryption key. Otherwise try another guess of G.

Again the majority of the computational complexity is taken by the guessing of G and computing $R_3^{-1}(ct, K_2) + W_G(K_2) + E_G^2(K_{32})$ for each guess of $K_2 \cup K_{32}$ and $L_G(v) + Q_G(K_1 + E_G^1(K_{31})) + C_G$ for each guess of $K_1 \cup K_{31}$. This part takes $2^{s \cdot r_2} \cdot 2^{1+n/2}$. If $r_1 + r_3 = \lfloor \frac{n}{3s} \rfloor - \Delta$ then the complexity can be rewritten as $2^{s \cdot r + s \cdot \Delta - n/3 + n/2} = 2^{sr + s\Delta + n/6}$. For $r = 0.8\lfloor \frac{n}{s} \rfloor$, this complexity is around $2^{29n/30 + s\Delta}$. Thus the procedure becomes a valid attack if and only if $s\Delta < n/30$. Thus since Δ is at least 1 when the first and last keybits are not all linearly independent, the procedure does not work for all challenge instances when $s = 10$.

6 Improving Complexities using the 3-xor problem

The 3-xor problem in a nutshell is as follows: given 3 lists $L_1,\ L_2,\ L_3$ of binary strings over $\{0,1\}^n$, the task is to find 3 elements $x_1 \in L_1,\ x_2 \in L_2,\ x_3 \in L_3$ such that $x_1 \oplus x_2 \oplus x_3 = 0$. This problem has been extensively studied in the literature. Wagner studied in [Wag02], the generalized k-xor problem and showed that for the 4-xor problem if we have lists of size $2^{n/3}$ then a solution can be found in time $O(2^{n/3})$. However the 3-xor problem

still required $O(2^{n/2})$ time using his approach. In [Nan15] a forgery attack was mounted against the COPA mode of operation requiring only $2^{n/3}$ encryption queries and about $2^{2n/3}$ time. This attack was later refined in [NS15], using an improved 3-xor algorithm, to $2^{n/2-\epsilon}$ queries and $2^{n/2-\epsilon}$)operations, for small ϵ. In [LS19], the authors attacked the 2-round Even Mansour algorithm using this problem with data and time both lower than 2^n. However, the algorithm we use was proposed by Joux [Jou09, Section 8.3.3.1], which is the best algorithm for the 3-xor problem to this day. A generalization for the above algorithm for variable sized lists was proposed in [BDF18], however since we will use lists of fixed size in this section, Joux's algorithm is more relevant here.

Before we discuss the details of the attack it is best to summarize the algorithm in a few words. We begin with the following lemma.

Lemma 2. *Given $n/2$ randomly generated vectors over $\{0,1\}^n$, then with high probability they are linearly independent.*

Proof. The above probability is given by $p = 2^{-n^2/2} \cdot \prod_{i=0}^{n/2-1}(2^n - 2^i)$. For large n, this equals

$$p = \prod_{i=0}^{n/2-1}\left(1 - \frac{2^i}{2^n}\right) \approx 1 - \frac{\sum_{i=0}^{n/2-1} 2^i}{2^n} = 1 - \frac{2^{n/2}-1}{2^n} \approx 1 - 2^{-n/2}$$

□

The algorithm proceeds with 3 lists $L_1,\ L_2,\ L_3$ of size $2^{n/2}/\ell,\ 2^{n/2}/\ell,\ \ell^2$ respectively where $\ell = \sqrt{n/2}$. The list L_3 has $n/2$ random vectors which span at most a subspace of rank $n/2$. It is possible to choose vectors $\mathbb{B} = \{b_1, b_2, \ldots, b_{n/2},\ b_{n/2+1}, b_{n/2+2}, \ldots, b_n\}$ such that all vectors in L_3 belong to the subspace generated by $b_{n/2+1}, b_{n/2+2}, \ldots, b_n$. Now designate M to be the $n \times n$ binary matrix that changes the basis of all vectors in $L_1,\ L_2,\ L_3$ to $\mathbb{B}$. Note that in the modified basis all elements in L_3 will begin with $n/2$ zeros. From the previous lemma, we know that the elements in L_3 are linearly independent with very high probability. In that case $b_{n/2+1}, b_{n/2+2}, \ldots, b_n$ can be simply taken as the elements of L_3 which ensures that in the modified basis the elements of L_3 have hamming weight exactly equal to 1, i.e. it has 1 in one of the positions from $n/2+1$ to n. Note that if there exists 3 vectors $x_1 \in L_1,\ x_2 \in L_2,\ x_3 \in L_3$ such that $x_1 \oplus x_2 \oplus x_3 = 0$, then $Mx_1 \oplus Mx_2 \oplus Mx_3 = 0$ for any $n \times n$ binary matrix M. Once M is fixed, it can be used to transform L_1 and L_2. After this, all we need to do is to search for pairs of elements $(x_1, x_2) \in L_1 \times L_2$ such that $M \cdot x_1 \oplus M \cdot x_2$ equals 0 on the first $n/2$ bits, (and when all the vectors in L_3 are linearly independent we simply have to check if the sum has hamming weight 1) and this of course can be done efficiently in the following way.

1. After transforming all elements of L_3 in the new basis $\mathbb{B}$, insert the elements in hash table J.

2. After transforming all elements of L_1 in the new basis $\mathbb{B}$, insert the elements in hash table H indexed by first $n/2$ bits. All cells of the table should be able to hold multiple elements.

3. After transforming all elements of L_2 in the new basis $\mathbb{B}$, insert the elements in the same hash table H indexed by first $n/2$ bits. Note that if any cell of H has more than one elements then their sum in the first $n/2$ bits must be 0. By standard randomness assumptions there will be $\frac{2^{n+1} \cdot 2^{-n/2}}{n} = \frac{2^{n/2+1}}{n}$ such pairs left whose sum needs to be tested for membership in L_3.

4. Assuming that testing for membership in J can be done in constant time, we need $\frac{2^{n/2+1}}{n}$ tests. Note that most of the time L_3 is linearly independent and so testing for membership in L_3 can be done by simply checking whether the hamming weight of

the full vector is 1, and whether it begins with $n/2$ zeros. In this case, it is neither necessary to change the basis of vectors in L_3 nor store them anywhere.

Since the complexity of preparing each list is $O(2^{n/2}/\sqrt{n/2})$ and around $O(2^{n/2+1}/n)$ membership tests are required the complexity of the algorithm is $O(2^{n/2+1}/\sqrt{n/2} + 2^{n/2+1}/n) \approx O(2^{n/2+1}/\sqrt{n/2})$. This gives a speedup of around $\sqrt{n/2}$ compared to the basic birthday algorithm of Wagner.

6.1 MITM on 2-round full S-box layer

The improved algorithm closely follows the one presented in Sec 5.1 earlier. The basic idea is still the same: this time we partition the key K into 3 sets $K_1 = \{k_0, k_1, k_2, \ldots, k_{m-1}\}$, $K_2 = \{k_m, k_{m+1}, k_{m+2}, \ldots, k_{2m-1}\}$ and $K_3 = \{k_{2m}, k_{2m+1}, \ldots, k_{n-1}\}$, where the value of m is given by $\lfloor \log_2(2^{n/2}/\sqrt{n/2}) \rfloor$, and so the size of K_3 is considerably smaller and only around $\lfloor \log_2(n/2) \rfloor$.

Our strategy will be, as before, to guess the s majority bits $\eta_1, \ldots, \eta_s$ at the input of the second round inverse S-boxes to linearize R_2 (and of course its inverse). Borrowing the terminology from Sec 5.1, where x denotes the n-bit input to the second round and RK_1, RK_2 denote the first, second round keys which are linear functions of the original key $K = RK_0$, we have $R_1(pt + RK_0, RK_1) = x$ and $R_2(x, RK_2) = ct$. Since after guessing the majority bits η_I the inverse of R_2 becomes linear, we can write each bit x_i of x as $x_i = A_i(K_1) + B_i(K_2) + C_i(K_3) + d_i,\ \forall\ i \in [0, n-1]$, where all A_i, B_i, C_i are linear functions and d_i is a constant.

Similarly in R_1, the set of keybits in K_1, K_2, K_3 can be partitioned in a manner so that they are not combined multiplicatively in the first round. Hence computing R_1 in the forward direction from the plaintext input it is possible to write each x_i as $f_i(K_1) + g_i(K_2) + h_i(K_3) + e_i,\ \forall\ i \in [0, n-1]$, where all f_i, g_i, h_i are quadratic functions and e_i is a constant. Equating these expressions we have $A_i(K_1) + B_i(K_2) + C_i(K_3) + d_i = f_i(K_1) + g_i(K_2) + h_i(K_3) + e_i$. Rearranging terms we have:

$$\underbrace{[A_i(K_1) + f_i(K_1) + d_i]}_{L_1} + \underbrace{[B_i(K_1) + g_i(K_1) + e_i]}_{L_2} + \underbrace{[C_i(K_3) + h_i(K_3)]}_{L_3} = 0$$

Note that if 3 lists are enumerated for the terms in the square braces, then we arrive exactly at the scenario of the 3-xor problem. We need to find 3 elements form these lists that sum to 0. So our modified algorithm will be as follows:

1. Calculate the functional forms of f_i, g_i, h_i and e_i for all $i \in [0, n-1]$.

2. Guess the values $\eta_1, \ldots, \eta_s$. This step is done 2^s times in the worst case.

   - Compute A_i, B_i, C_i, d_i for all $i \in [0, n-1]$ using the guessed values.
   - For all possible values of K_3, create a hash table L_3 indexed by the n-bit vector $[C_i(K_3) + h_i(K_3)],\ \forall\ i \in [0, n-1]$. From here find the matrix M that would transform basis the basis $\mathbb{B} = \{b_1, b_2, \ldots, b_{n/2},\ b_{n/2+1}, b_{n/2+2}, \ldots, b_n\}$ such that L_3 is spanned by $b_{n/2+1}, b_{n/2+2}, \ldots, b_n$. With high probability the list L_3 is linearly independent so that $b_{n/2+1}, b_{n/2+2}, \ldots, b_n$ can be taken to be the vectors in L_3. Multiply all vectors in L_3 by M and store in a hash table J. Note there are around $n/2$ steps here.
   - For all possible values of K_1, create a hash table L_1 indexed by the n-bit vector $M \cdot [A_i(K_1) + f_i(K_1) + d_i],\ \forall\ i \in [0, n-1]$. We need $2^{n/2}/\sqrt{n/2}$ operations in this step.

- For all possible values of K_2, create a hash table L_2 indexed by the n-bit vector $M \cdot [B_i(K_1) + g_i(K_1) + e_i],\ \forall\ i \in [0, n-1]$. We need $2^{n/2}/\sqrt{n/2}$ operations in this step.
- Note that in practice, 2 different hash tables are not necessary. We can instead use one single hash table H in which all elements of L_1, L_2 are inserted indexed by the first $n/2$ bits as explained in the previous subsection.
- For all pairs in $x_1, x_2 \in H$ which are in the same cell

  A: Discard if the sum is not in L_3. For most cases this can easily be verified by checking if the hamming weight of the sum is 1, i.e. if L_3 is linearly independent.
- Once a solution for $K_1,\ K_2$ and K_3 is found, check if the majority bits are consistent with the guess of the key. If yes, this key is in fact the encryption key. Otherwise try another guess of $\eta_1, \ldots, \eta_s$.

For each majority guess, the complexity of the attack is dominated by finding a collision between two lists of length $O(2^{n/2}/\sqrt{n/2})$. So the total complexity of the attack is $O(2^s \times 2 \cdot 2^{n/2}/\sqrt{n/2}) = O(n^{-1/2} \cdot 2^{s+n/2+1})$. This gives a speed up of around $\sqrt{n/2}$ over the attack in Section 5.1.

There are some further issues to be discussed. We ideally want the lists L_1 and L_2 of the same size, but it is often not possible due to the algebraic structure of LowMC. Since we have to partition the keybits such that the cardinality of each set should be a multiple of 3, it is not always possible to get lists of size $2^{n/2}/\sqrt{n/2},\ 2^{n/2}/\sqrt{n/2}$ and $n/2$. For $n = 129$ we have $n/2 = 64.5 \approx 2^6$, and so we can take K_3 to be the last 6 bits of the key, and K_1 and K_2 may contain the first 60 and the next 63 bits of the key respectively. In that case, the cost of preparing the lists is around $2^{60} + 2^{63} \approx 2^{63}$. The sum of the transformed vectors in L_1 and L_2 would need to be zero in the first $129 - 64 = 65$ bits and so after filtering $2^{60+63-65} \approx 2^{58}$ vector sums need to be tested for membership in L_3. So the total cost is around $2^{63} + 2^{60} + 2^{58} \approx 2^{63}$. Multiplying this by the 2^{43} times we need to guess majority bits, this comes to $2^{63+43} = 2^{106}$, which results in a speed up of factor 8 compared to the basic MITM in Section 5.1. For $n = 192$, we have $n/2 = 96 \approx 2^{6.58}$. The only feasible choice of the size of K_3 is again 6, which forces K_1 and K_2 to be of size 93 each. The cost of preparing lists is around $2^{93} + 2^{93} = 2^{94}$. However the number of pairs needed to be tested for membership in L_3 is $2^{93+93-(192-64)} = 2^{58}$. So the total complexity for list matching is around $2^{94} + 2^{58} \approx 2^{94}$. Multiplying by the number of majority guesses, we get the total complexity as $2^{64+94} = 2^{158}$ which also results in a speed up of 8 compared to Section 5.1. Similarly for $n = 255$, we have to take K_1, K_2, K_3 of sizes $123, 126, 6$ respectively. A similar calculation yields the total complexity as $2^{85+126} = 2^{211}$ which results again in a speedup of 8 compared to the basic MITM.

7 Improved MITM attack on 2-rounds with full S-box layer

Let us revisit the attack in Sec 5.1. After guessing the majority bits of the second round and linearizing it, we have already seen that the algebraic relation between the plaintext and ciphertext can be written as

$$f_i(K_1) + A_i(K_1) + c_i = g_i(K_2) + B_i(K_2) + d_i,\ \forall\ i \in [0, n-1] \tag{1}$$

Note that the functions $A_i,\ B_i$ are linear and $f_i,\ g_i$ are quadratic. Not only that, $f_i,\ g_i$ can be expressed as a affine function in an extension of the input of double size. This comes from the structure of the Sbox: $S(x_0, x_1, x_2)$ is an affine function on $(x_0, x_1, x_2, x_0x_1, x_1x_2, x_2x_0)$.

Let $\overline{f}_i, \overline{g}_i$ be the affine functions associated with $f_i,\ g_i$. Therefore the above set of equations can be written as

$$\overline{f}_i(\overline{K}_1) + A_i(\overline{K}_1) + c_i + d_i = g_i(K_2) + B_i(K_2),\ \forall\ i \in [0, n-1] \tag{2}$$

where if $K_1 = [k_0, k_1, k_2, \ldots, k_{3w-3}, k_{3w-2}, k_{3w-1}]$, we define $\overline{K}_1 = [k_0, k_1, k_2, k_0k_1, k_1k_2, k_2k_0, \ldots, k_{3w-3}, k_{3w-2}, k_{3w-1}, k_{3w-3}k_{3w-2}, k_{3w-2}k_{3w-1}, k_{3w-1}k_{3w-3}]$. Note that unlike in Sec 5.1, we are not splitting the key into equal halves: K_1 only has the first $3w$ bits of the master key.

Since $F_i = \overline{f}_i + A_i$ is an affine function over $\overline{K}_1$ the map $\phi : \overline{K}_1 \to [F_0, F_1, \ldots, F_{n-1}]$ can be seen as a linear code of length n and dimension $6w$. Let w be such that K_1 contains around $n/3$ keybits i.e. $w \approx n/9$ and hence K_2 contains the remaining $2n/3$ keybits. Since ϕ is seen as a linear code, let $\mathbf{G}$ be the corresponding generator matrix (of size $6w \times n \approx 2n/3 \times n$), which can be easily constructed from the algebraic forms of the functions F_i. Let $\mathbf{H}$ be the parity check matrix of the code (of size $n/3 \times n$). The parity check matrix is essentially obtained from the generator matrix by employing one gaussian elimination. Define Con to be the vector $[c_0 + d_0, c_1 + d_1, \ldots c_{n-1} + d_{n-1}]^T$. Note that the left side of Equation (2), when written in matrix notation for all $i = 0, 1, \ldots, n-1$ is essentially $\phi(\overline{K}_1) + Con$. Therefore we have $\mathbf{H} \cdot [\phi(\overline{K}_1) + Con] = \mathbf{H} \cdot [\mathbf{G}\overline{K}_1 + Con] = \mathbf{H} \cdot Con = e$ (say).

We can split K_2 into two halves K_{21} and K_{22} such that both halves contain approximately $n/3$ keybits each. We can rewrite $g_i(K_2) + B_i(K_2)$ as $g_i^1(K_{21}) + B_i^1(K_{21}) + g_i^2(K_{22}) + B_i^2(K_{22})$ for all $i \in [0, n-1]$, where g_i^j are quadratic and B_i^j are linear for $j = 1, 2$. Again this is possible due to the structure of LowMC in which quadratic terms from adjacent S-boxes do not combine multiplicatively after one round. Define the n-bit vectors $M_1 = [g_0^1(K_{21}) + B_0^1(K_{21}), \ldots, g_{n-1}^1(K_{21}) + B_{n-1}^1(K_{21})]^T$ and $M_2 = [g_0^2(K_{22}) + B_0^2(K_{22}), \ldots, g_{n-1}^2(K_{22}) + B_{n-1}^2(K_{22})]^T$. Multiplying both sides of Eqn (2) by $\mathbf{H}$ for $i = 0, 1, \ldots, n-1$, we get the matrix equation:

$$\mathbf{H} \cdot (M_1 + M_2) = e,\ \ \Rightarrow \mathbf{H} \cdot M_1 = \mathbf{H} \cdot M_2 + e$$

Double MITM strategy: Note that M_1 and M_2 only contain expressions on the keybits in the sets K_{21} and K_{22} respectively. Thus we can conduct a first MITM stage in which we create 2 lists L_1, L_2. L_1 contains the values $\mathbf{H} \cdot M_1$ for all $2^{n/3}$ values of K_{21}. This amounts to $2^{n/3} \cdot [n/3]$ bits of memory. And similarly the list L_2 contains the values $\mathbf{H} \cdot M_2 + e$ for all $2^{n/3}$ values of K_{22}. We look for a collision in the $n/3$ co-ordinates of these lists. We are expected to get around $2^{n/3+n/3-n/3} \approx 2^{n/3}$ collisions. Thus in the process we get $2^{n/3}$ key values for the keybit set $K_2 = (K_{21}, K_{22})$.

2nd MITM: Let us now turn to Eqn (1). The left side of this equation is defined over the $n/3$-bit set K_1 which can have $2^{n/3}$ values in total. And we have just reduced K_2 to a set of $2^{n/3}$ values. Thus the next MITM is making two more lists $L_3,\ L_4$ of size $2^{n/3}$ each in the following way. L_3 contains all $2^{n/3}$ values $[f_i(K_1) \oplus A_i(K_1) \oplus c_i],\ \forall\ i \in [0, n-1]$ enumerated for all the $2^{n/3}$ values of K_1. For all the $2^{n/3}$ values of K_2 that have passed the previous MITM step we make the list L_4 containing $[g_i(K_2) \oplus B_i(K_2) \oplus d_i],\ \forall\ i \in [0, n-1]$. We now look for a collision between L_3 and L_4. On average we have $2^{n/3+n/3-n} \approx 2^{-n/3}$ collisions. Note that the correct key K will necessarily by the output of one of this MITM step for the correct guess of majority bits in the second round. We are now ready to state the attack formally.

1. Calculate the functional forms of $f_i, g_i, \overline{f}_i, g_i^1, g_i^2$ and c_i for all $i \in [0, n-1]$.

2. Guess the majority values $\eta_1, \ldots, \eta_s$ at the output of 2nd round S-box layer as in Sec 5.1. This step is done 2^s times in the worst case (note $s = n/3$).

- Compute A_i, B_i, d_i for all $i \in [0, n-1]$ using the guessed values.
- Compute the functions $F_i = \overline{f}_i + A_i$ for all $i \in [0, n-1]$.
- Using the F_i's, construct the generator matrix $\mathbf{G}$.
- Using gaussian elimination, construct the parity check matrix $\mathbf{H}$.
- Construct $Con = [c_0 + d_0, c_1 + d_1, \ldots c_{n-1} + d_{n-1}]^T$, and $e = \mathbf{H} \cdot Con$.
- For all possible values of K_{21}, create a hash table L_1 indexed by the $n/3$-bit vector $\mathbf{H} \cdot M_1$. We need $2^{n/3}$ operations in this step.
- For all possible values of K_{22}, create a hash table L_2 indexed by the $n/3$-bit vector $\mathbf{H} \cdot M_2 + e$. We need $2^{n/3}$ operations in this step.
- Find all collisions between L_1 and L_2. Store all values of K_{21}, K_{22} extracted from the collision in a list L.
- For all possible values of K_1, create a hash table L_3 indexed by the n-bit vector $[f_i(K_1) \oplus A_i(K_1) \oplus c_i],\ \forall\ i \in [0, n-1]$. We need $2^{n/3}$ operations in this step.
- For all values of $K_2 \in L$, create a hash table L_4 indexed by the n-bit vector $[g_i(K_2) \oplus B_i(K_2) \oplus d_i],\ \forall\ i \in [0, n-1]$. We need $2^{n/3}$ operations in this step.
- When a collision is found for K_1 and K_2 check if the majority bits are consistent with the guess of the key. If yes, this key is in fact the encryption key. Otherwise try another guess of $\eta_1, \ldots, \eta_s$.

Complexity Estimation: For each guess of $2^{n/3}$ majority values, we have to perform a gaussian elimination and 2 MITM steps each of complexity $2 \cdot 2^{n/3}$. The first MITM only requires evaluation of linear expressions while the second requires evaluation of quadratic expressions as well. Assuming that both type of expressions take similar time to evaluate, the total time complexity for this attack is $2^{n/3}$ gaussian eliminations + $2^{2+n/3+n/3} \approx 2^{2n/3+2}$ expression evaluations. For $n = 129$ this comes to 2^{88} evaluations.

8 Improved MITM attack on partial S-box layers

To mount this attack let us split the LowMC into 4 parts as shown in Fig 6:

1. First $a + b$ rounds which have been transformed as per the description given in Fig 3.
2. Final c rounds which have been transformed as per the description given in Fig 4.
3. The remaining $d = r - a - b - c$ rounds which lie in between.

Let the set of roundkey bits in the first a, b and the last c rounds be denoted as $K_a = [\kappa_0, \kappa_1, \ldots, \kappa_{3sa-1}]$, $K_b = [\kappa_{3sa}, \kappa_{3sa+1}, \ldots, \kappa_{3sa+3sb-1}]$ and $K_c = [\kappa_{n-3sc}, \kappa_{n-3sc+1}, \ldots, \kappa_{n-1}]$ respectively. Denote by K_{rem} the remaining $n-3s(a+b+c)$ key bits such that K_a, K_b, K_c, and K_{rem} are linearly independent expressions of the master key and so any key bit can be expressed as a linear function of them. Let $X = [x_0, x_1, x_2, \ldots, x_{n-1}]$ be the output of the first a rounds, $W = [\omega_0, \omega_1, \ldots, \omega_{n-1}]$ be the output of the first $a + b$ rounds and $Y[= y_0, y_1, \ldots, y_{n-1}]$ be the input to the last c rounds as shown in Fig 6. Let us look at the middle b and $d = r - a - b - c$ rounds closely as seen in Figure 7. Let us introduce $6b \cdot s$ new variables $U = [u_0, u_1, \ldots, u_{3bs-1}]$ and $Z = [z_0, z_1, \ldots, z_{3bs-1}]$ such that they represent the input and output bits of the $b \cdot s$ S-boxes in the middle b rounds. Our fist aim is to find a linear expression relating the x_i's, y_i's and z_i's and the keybits.

Let $D = [D_0, D_1, \ldots, D_{n-1}]$ be the output of the first of the b rounds (see Fig 7). Then we can write $D = Lin_1(z_0, z_1, \ldots, z_{3s-1}, x_{3s}, x_{3s+1}, \ldots, x_{n-1})$, where Lin_1 denotes a set of n affine functions. Similarly, if $E = [E_0, E_1, \ldots, E_{n-1}]$ is the output of the next round

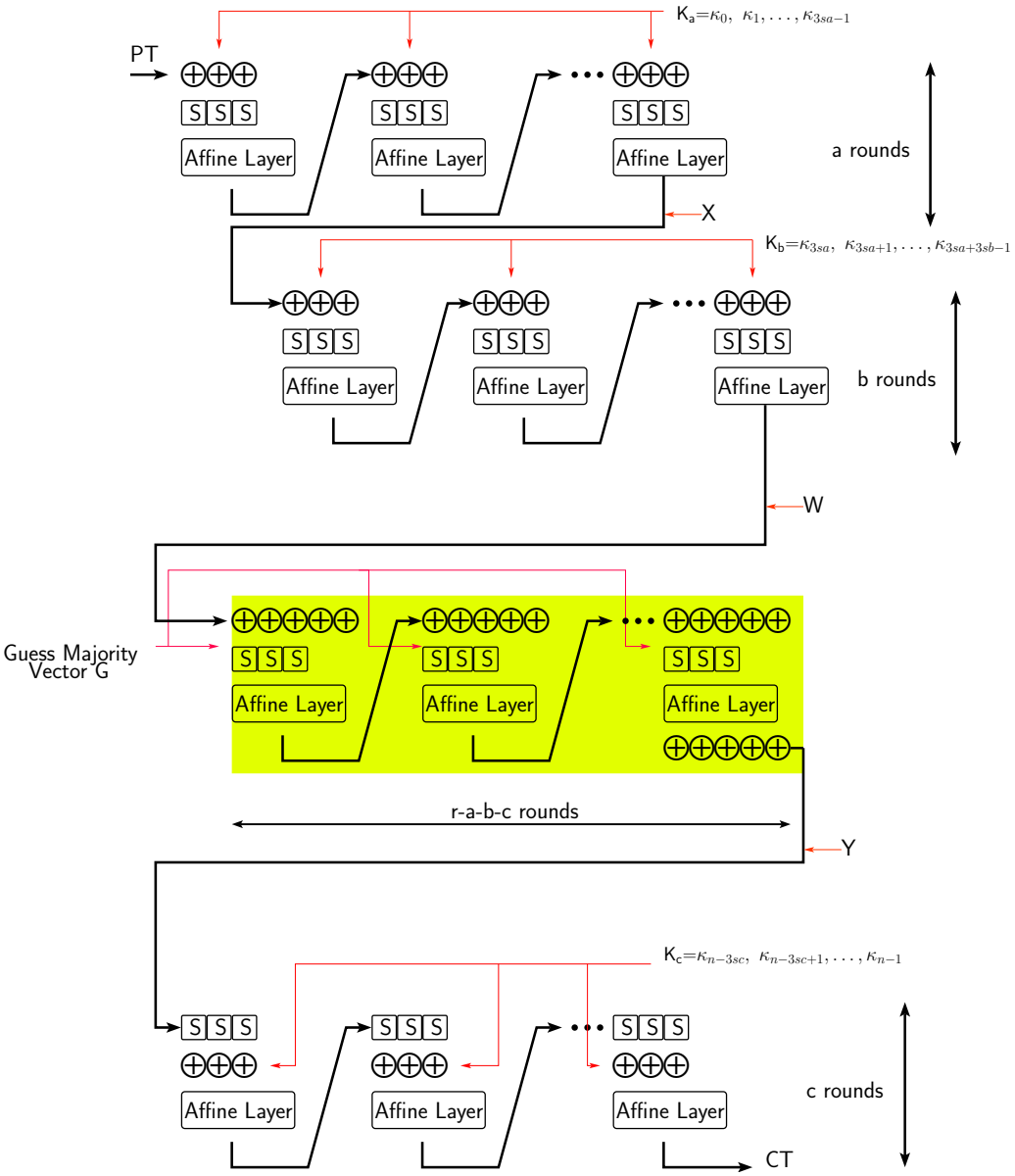

Figure 6: Splitting LowMC into 4 sections

we can write E as a set of linear functions on $(z_{3s}, z_{3s+1}, \ldots, z_{6s-1}, D_{3s}, D_{3s+1}, \ldots, D_{n-1})$ which means that we can write $E = Lin_2(z_0, z_1, \ldots, z_{6s-1}, x_{3s}, x_{3s+1}, \ldots, x_{n-1})$ as a set of linear functions on X and the first $6s$ z_i's. Iterating upto all b rounds, it can be seen that W can be written as a set of linear functions on the entire Z and $x_{3s}, x_{3s+1}, \ldots, x_{n-1}$. Now if we guess the majority bits at the inputs of the following d rounds, they become completely linear. In that case Y itself becomes linear in W and K_a, K_b, K_c, K_{rem} (since the key bits used in these d-rounds can be seen as linear expressions in K_a, K_b, K_c, K_{rem}). Hence we have $Y = Lin(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem})$. The above equation denotes a system of n affine equations (one for each bit in Y) in all the n bits of the Key. Our aim is to get a reduced set of equations by somehow eliminating Z, K_b, K_{rem} from this set. Note that the set $\Lambda = \{Z, K_b, K_{rem}\}$ comprises a total of $\theta = 3sb + 3sb + (n - 3s(a + b + c))$ variables. Consider the system of n equations $Y = Lin(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem})$. Apart from the θ variables the system has n (for Y) $+ n - 3s$ (for X) $+ 6s$ (for K_a, K_c) $= 2n + 3s$ variables. So the above system can be written in matrix notation as $\mathbb{M} \cdot \mathbf{v} = 0$, where $\mathbf{v}$ is the set of $2n + 3s + \theta$ variables and $\mathbb{M}$ is a matrix over GF(2) of size $2n + 3s + \theta \times n$. If we rearrange $\mathbf{v}$ so that the variables in Λ are the first θ elements of $\mathbf{v}$ then a simple gaussian elimination that sweeps out at least the first θ columns of $\mathbb{M}$ is sufficient for this purpose. The last $n - \theta$ rows of the matrix would then have the entries in the first θ columns all equal to 0 and thus these are the linear equations in K_a, K_c, X, Y that we get from this process. Note we have a total of $n - \theta = 3sa + 3sc - 3sb$ equations of this form.

First MITM: The equations so obtained can be rearranged and written as $Aff_1(K_a, X) = Aff_2(K_c, Y)$, where Aff_1, Aff_2 are a set of $3sa + 3sc - 3sb$ affine functions on K_a, X and K_c, Y respectively. We now state the first MITM step: note that if we guess the value of K_a, we can easily obtain the value of X by computing the forward a rounds from the plaintext. If we guess K_c we can similarly compute Y, by computing backward the last c

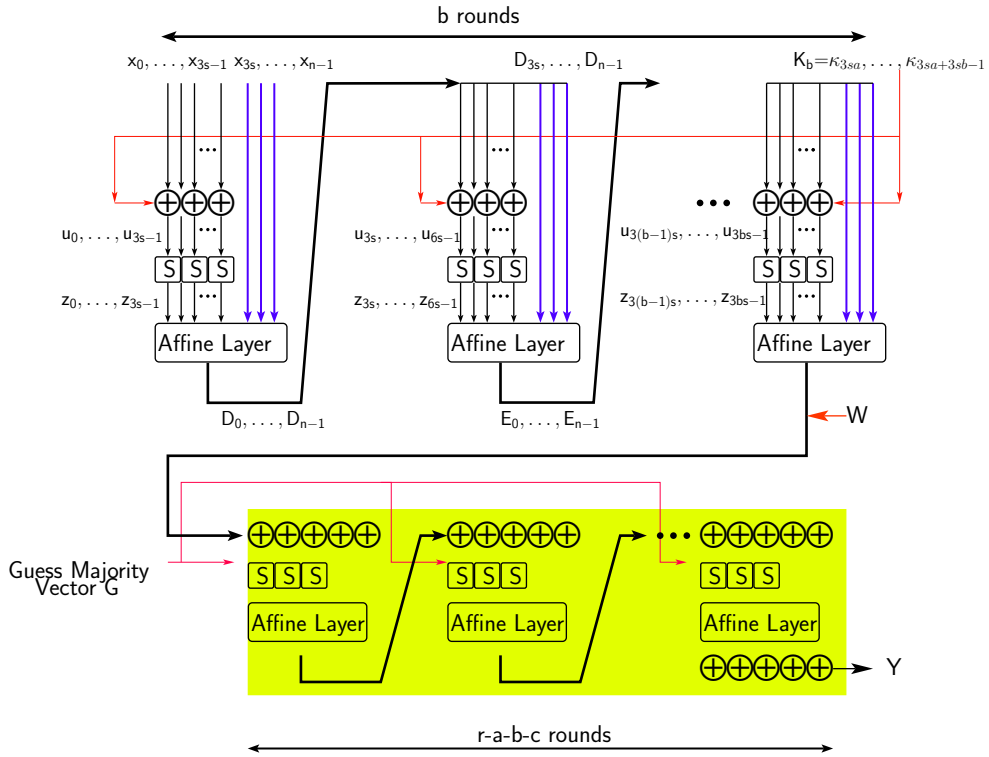

Figure 7: The middle $b + d$ rounds

rounds from the ciphertext. Hence for all the 2^{3sa} values of K_a we make the first list L_1 that contains all calculated values of $Aff_1(K_a, X)$. Similarly for all the 2^{3sc} values of K_c we make the second list L_2 that contains all calculated values of $Aff_2(K_c, Y)$. We look for collisions in the two lists. We can expect around $2^{3sa+3sc-(3sa-3sb+3sc)} = 2^{3sb}$ collisions. We store all the 2^{3sb} tuples (K_a, K_c) so obtained in a list L.

Second MITM: The second part of the attack focuses on getting an affine relation between U, Z and K_b. From Figure 7, we can see that $u_i = x_i + \kappa_{3sa+i}$, $\forall i \in [0, 3s-1]$. For the second round we have

$$\begin{aligned} u_{3s+i} &= D_i + \kappa_{3sa+3s+i},\ \forall\ i \in [0, 3s-1] \\ &= lin(z_0, \ldots, z_{3s-1}, x_{3s}, \ldots, x_{n-1}) + \kappa_{3sa+3s+i},\ \forall\ i \in [0, 3s-1] \end{aligned}$$

where lin is a linear function. The above holds since we have already seen that all D_i's are linear functions in $(z_0, \ldots, z_{3s-1}, x_{3s}, \ldots, x_{n-1})$. Similarly for the third round we have

$$\begin{aligned} u_{6s+i} &= E_i + \kappa_{3sa+6s+i},\ \forall\ i \in [0, 3s-1] \\ &= lin_1(z_0, \ldots, z_{6s-1}, x_{3s}, \ldots, x_{n-1}) + \kappa_{3sa+6s+i},\ \forall\ i \in [0, 3s-1] \end{aligned}$$

where lin_1 is another linear function. Iterating over all the b rounds we can write, $U = K_b + P(Z, x_{3s}, \ldots, x_{n-1})$, where P denotes a set of $3bs$ linear expressions. We can now replace K_b in our original set of equations $Y = Lin(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem})$ to get

$$\begin{aligned} Y &= Lin(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, U + P(Z, x_{3s}, \ldots, x_{n-1}), K_c, K_{rem}) \\ &= Lin'(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, U, K_c, K_{rem}) \end{aligned}$$

This time we want to eliminate K_{rem} from the above set of linear equations using the same gaussian elimination method as in the previous stage. There are $n - 3s(a+b+c)$ variables in K_{rem} that we eliminate, which leaves us with $3s(a+b+c)$ equations in $Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, U, K_c$. We can rearrange the terms in the equation to get $Aff_3(Z, U) = Aff_4(X, K_a, K_c)$, where Aff_3, Aff_4 are a set of $3s(a+b+c)$ affine functions on Z, U and K_a, K_c, X respectively. Note that if we guess Z, we can compute U since the S-box is bijective, and we have already seen that guessing K_a lets us compute X by computing the a forward rounds from the plaintext. Thus in the next MITM stage we make 2 lists L_3, L_4. In L_3 we store the $3s(a+b+c)$-bit vector given by the expressions $Aff_3(Z, U)$ for each of 2^{3bs} values of Z. In L_4 we store the $3s(a+b+c)$-bit vector given by the expressions $Aff_4(X, K_a, K_c)$ for each of 2^{3bs} values of (K_a, K_c) in L. We again look for collisions in the 2 lists. The expected number of collisions is $2^{3bs+3bs-3s(a+b+c)} = 2^{3sb-3sa-3sc}$. Again the correct value of the key K_a, K_c is guaranteed to be the outcome of the collision finding stage for the correct guess of the majority values.

Once we get a candidate solution K_a, K_c, Z, U we can compute the vectors X, Y by computing the a, c rounds forwards/backwards from the plaintext/ciphertext. We can then compute $K_b = U + P(Z, x_{3s}, \ldots, x_{n-1})$. As we know the majority of the inputs of the S-boxes in $r - a - b - c$ middle rounds, we can solve an affine equation of form $\mathsf{Aff}_{rem}(W, K_{rem}) = Y$ to recover the value of K_{rem}, which was the only part of the key which remained unknown. After this one can check if the key so obtained produces the required majority values guessed at the beginning. If not the attacker can re start the process with another set of majority values. The expected number of such checks is around $2^{s(r-a-b-c)+3sb-3sa-3sc} = 2^{rs-4sa-4sc+2sb}$.

We continue by formally stating the attack.

1. Separate the first $a+b$ and last c rounds of the cipher
2. Denote the output of the first a rounds by X, the output of the b rounds by W and the input of the last c rounds by Y.

3. Denote the inputs/outputs of the S-boxes in the b rounds by U/Z
4. Guess the majority bits of the inputs of the S-boxes of $r-a-b-c$ middle rounds.
5. For every Majority guess do:

   First MITM:

   - Compute the relation $Y = Lin(Z, x_{3s}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem})$
   - Eliminate K_b, K_{rem}, Z from the relation and form and equation of form $Aff_1(K_a, X) = Aff_2(K_c, Y)$.
   - By exhausting all possible values of K_a keep a list of $Aff_1(K_a, X)$, where X is computed knowing K_a and plaintext pt.
   - Try all possible values of K_c and find collisions between $Aff_2(K_c, Y)$ and the list computed in the previous step. Keep a list of (K_a, K_c) values satisfying the condition.

   Second MITM:

   - Compute the relation $Y = Lin'(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, U, K_c, K_{rem})$ by replacing K_b.
   - Eliminate K_b, K_{rem} to get a relation of form $Aff_3(Z, U) = Aff_4(X, K_a, K_c)$.
   - For every pair (K_a, K_c) in the list computed in first MITM, compute $Aff_4(X, K_a, K_c)$.
   - For every possible value of Z, compute $Aff_3(Z, U)$, where U can be computed efficiently from Z, and look for occurrence with $Aff_3(Z, U)$ in the list from the previous step.
   - For every (K_a, K_c, Z, U) satisfying the relation, compute K_b, W, Y as shown before.
   - Linearize the middle $r-a-b-c$ rounds using the majority guess and compute K_{rem} from $\mathsf{Aff}_{rem}(K_{rem}, K_a, K_b, K_c, W) = Y$.
   - After the entire key is found, check if they result in the same majority values assumed at the beginning of the attack or else retry with another set of majority values.

Complexity Estimation: Both MITM steps should be done for each majority guess for the middle rounds, hence should be repeated $2^{s(r-a-b-c)}$ times. Computing the relations is done by propagating some round functions so can be done in less time than an encryption. Eliminating variables is done by a Gaussian elimination like process, similar to a Gaussian elimination on a $(3sa + 3sc - 3sb) \times (3sa + 3sc - 3sb)$ matrix.

The first MITM takes time less than $2^{3sa} + 2^{3sb}$ encryptions. The number of pairs stored in the first MITM is around 2^{3sb} as mentioned before.

Later on we replace K_b in the Linear equation and eliminate K_b, K_{rem}, this can also be seen as a matrix multiplication followed by a Gaussian elimination. Next we compute the values of $Aff_3(Z, U)$ and $Aff_4(X, K_a, K_c)$ having values of K_a, K_c and Z. Computing the value of U from Z takes a small constant number of operations and computing these values takes less than $2^{3sb} + 2^{3sb}$ encryptions. The expected number of collisions in this procedure is $2^{3sb-3sa-3sc}$ and for each of these collision pairs the attacker should run a Gaussian elimination for a matrix of size $(n - 3s(a+b+c)) \times (n - 3s(a+b+c))$. Hence the total complexity of the attack is less than $2^{s\times(r-a-b-c)} \times (2^{3sa} + 3 \times 2^{3sb})$ encryptions.

For each guess of majority values we need to do 2 Gaussian eliminations. So the number of gaussian eliminations required are $2^{s\times(r-a-b-c)+1}$. For instance by considering $a = b = c = r/3$ the attack requires 2^{sr+2} encryptions and only 2 Gaussian eliminations,

which would successfully break the cipher when the number of rounds is less than $\lfloor \frac{n}{s} \rfloor - 2$. Note that when $a = b = c = r/3$, d becomes 0 which means that we do not need to guess any majority values in the middle d rounds to linearize the circuit. When this happens the part shaded in yellow in Figure 6 essentially reduces to one layer of key addition of n-bits.

9 Conclusion

In this paper we describe attacks on instances of LowMC where the number of S-boxes is less than the security level, when we use only one plaintext/ciphertext pair. A cryptanalysis of this kind is important as it results in a forgery on the post-quantum signature scheme PICNIC. Since our attacks are in the KPA/KCA scenario and since we use only one plaintext/ciphertext pair, it is not possible to apply traditional symmetric cryptanalytic techniques like differential, linear or any other higher order differential attacks. We begin by showing how to efficiently linearize the LowMC S-box by guessing only one single balanced quadratic expression in its input bits. We leverage this fact to present two types of attacks. First is a simple linearization attack where the attacker obtains a set of linear equations on the key bits relating the plaintext and ciphertext. The second is a meet in the middle attack, which takes advantage of the fact that in a single LowMC round, all key bits are not combined multiplicatively. We then show how to improve the attack on the 2-round full S-box layer variant of LowMC with the help of Joux's algorithm to solve the 3-xor problem. in the next sections we show how we can use a 2-stage MITM to improve attack complexities further.

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.

[BDF18] Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving algorithms for the 3xor problem. *IACR Trans. Symmetric Cryptol.*, 2018(1):254–276, 2018.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842, 2017.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-order cryptanalysis of lowmc. In *Information Security and Cryptology - ICISC 2015 - 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers*, pages 87–101, 2015.

[DKRS] Christoph Dobraunig, Daniel Kales, Chistian Rechberger, and Markus Schofnegger. Survey of key-recovery attacks onlowmc in a single plaintext/ciphertext scenario. https://raw.githubusercontent.com/lowmcchallenge/lowmcchallenge-material/master/docs/survey.pdf.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30, 2007.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. CRC Press, 2009.

[LIM20] Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques. *IACR Cryptol. ePrint Arch.*, 2020:1034, 2020.

[LS19] Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-mansour using the 3-xor problem. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, pages 210–235, 2019.

[Nan15] Mridul Nandi. Revisiting security claims of XLS and COPA. *IACR Cryptol. ePrint Arch.*, 2015:444, 2015.

[nis] Nist post quantum cryptography project. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[NS15] Ivica Nikolic and Yu Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 683–703, 2015.

[RST18] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of low-data instances of full lowmcv2. *IACR Trans. Symmetric Cryptol.*, 2018(3):163–181, 2018.

[Wag02] David A. Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.