New Attacks on LowMC instances with a Single Plaintext/Ciphertext pair

Subhadeep Banik, Khashayar Barooti, Serge Vaudenay and Hailun Yan

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

Abstract. Cryptanalysis of the LowMC block cipher when the attacker has access to a single known plaintext/ciphertext pair is a mathematically challenging problem. This is because the attacker is unable to employ most of the standard techniques in symmetric cryptography like linear and differential cryptanalysis. This scenario is particularly relevant while arguing the security of the PICNIC digital signature scheme in which the plaintext/ciphertext pair generated by the LowMC block cipher serves as the public (verification) key and the corresponding LowMC encryption key also serves as the secret (signing) key of the signature scheme. In the paper by Banik et al. (IACR ToSC 2020:4), the authors used a linearization technique of the LowMC S-box to mount attacks on some instances of the block cipher. In this paper, we first make a more precise complexity analysis of the linearization attack. Then, we show how to perform a 2-stage MITM attack on LowMC. The first stage reduces the key candidates corresponding to a fraction of key bits of the master key. The second MITM stage between this reduced candidate set and the remaining fraction of key bits successfully recovers the master key. We show that the combined computational complexity of both these stages is significantly lower than those reported in the ToSC paper by Banik et al.

1 Introduction

The LowMC family of block ciphers was first proposed by Albrecht et al. in [ARS+15] and was designed specifically for use in FHE and MPC applications due to its low multiplicative complexity. The block cipher uses a 3-bit S-box which is the only non-linear transformation in the construction. Both the linear layers and round key generation are done by multiplying with full rank matrices over $GF(2)$ of appropriate dimensions. The designers propose several instances of the block cipher, 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.

Recently, LowMC has been used in the PICNIC digital signature scheme in the following way. Let $E(K, pt)$ be the LowMC encryption of the plaintext pt using the key K. The plaintext/ciphertext pair $(pt, ct = E(K, pt))$ 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 plaintext/ciphertext pair (pt, ct) 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.

1.1 Previous Work

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. Dinur et al. [DLMW15] showed that around 2^{-38} fraction of its 80-bit key instances could be broken 2^{23} times faster than exhaustive search. Moreover, all instances that claimed to provide 128-bit security could be broken about 1000 times faster. In [DKP+19], the authors showed that for the LowMC instances that employs partial linear layers, each instance belonged to a large class of equivalent instances that differ in their linear layers. This led to a more efficient implementation of the cipher that required reduces the evaluation time and storage of computing the linear layers. 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. In [DN19] the authors proposed multi-target attacks on the PICNIC signature scheme. For a survey of key recovery attacks on LowMC, readers may check the survey done by Rechberger et al. [GKRS]. 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.

The LowMC cryptanalysis challenge asked for cryptanalysis of several instances of LowMC (in which the blocksize and keysize are equal), with both partial and complete non-linear layers given only one plaintext and ciphertext pair. In [BBDV20], some instances of the challenge were successfully solved. 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 fact, they showed various attacks on

A 2-round LowMC with complete non-linear layers.
B $0.8 \cdot \lfloor \frac{n}{s} \rfloor$-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.

The authors in [BBDV20] report attack complexities in number of linear/quadratic expression evaluations. However it is always preferable to have computational complexity reported in terms of number of encryptions. We actually show in this paper that the best complexity of these attacks are equivalent to $\frac{n}{2r} \times 2^{rs}$ encryptions (r denotes the number of rounds used in the encryption), as will be discussed later in this paper. In [BBDV20], the authors then presented a speedup of a factor of 8 over the MITM attack by using the 3-xor problem.

In [Din], the authors showed an ingenious method of finding roots of multiple polynomial systems over $GF(2)$. The n variables of the equation system are partitioned into two disjoint sets $y = y_0, y_1, \ldots, y_{m-1}$ and $z = z_0, z_1, \ldots, z_{p-1}$ (with $n = m + p$). It is argued that any random linear combination of the polynomials in the original equation system, has an isolated solution with high probability, i.e. if $(\hat{y}, \hat{z})$ is an isolated solution then $(\hat{y}, z')$ is not a solution for all $z' \neq \hat{z}$. The authors then observed that all such isolated solutions could be recovered bit-by-bit by computing $p+1$ partial sums for each candidate solution $\hat{y} \in \{0,1\}^m$. The first step is to randomly combine the original equation system into a system with smaller number of equations whose solutions can be found by brute force. These solutions are then used to compute partial sums and construct a candidate solution of the original equation system. This generic method of solving equations works quite well if the algebraic degree of the system is small and so it was applied to attack 3, 4 and 5 round LowMC with complete non-linear layers for some specific block-lengths. However, the method can not be applied to LowMC instances with partial non-linear layers, since the number of rounds in such instances are generally much higher, and the degree of the internal state variables (as a function of the key) doubles every round. [LIM21] reports an algebraic attack on LowMC. However the authors use the $n^{2.8}$ estimate (ignoring constant factors) to solve Gaussian elimination, to report the complexity of their attack. As such it is unclear if the complexity bounds they report are tight.

1.2 Contribution and Organization of the Paper

In this paper we present new improved attacks on LowMC instances that use the linearization technique of the LowMC S-box as a starting point. We first provide a more precise complexity analysis of the linearization attack and of its proof. Then, we present improved attacks on both **a)** the 2 and 3-round complete non-linear layer instance, and **b)** the $0.8 \cdot \lfloor \frac{n}{s} \rfloor$ and $\lfloor \frac{n}{s} \rfloor$-round LowMC instance with partial non-linear layers. We show that the attack complexity can be reduced if we perform the MITM in two separate stages: the first stage reduces the set of possible key candidates of a fraction of key bits to smaller set. A second MITM stage is then performed on this reduced candidate set and the candidates in the remaining fraction of the key bits. The paper shows how to efficiently formulate equations to perform the 2 MITM stages, and proves conclusively that the correct key can be found with certainty. It also shows that the combined computational complexity of the 2 attack stages is significantly lower than the complexities reported in [BBDV20]. Table 1 tabulates in detail the complexities

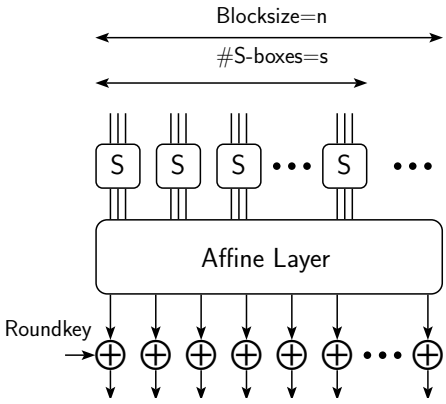


Fig. 1: LowMC Round Function

of the attacks reported in this paper and compares with the corresponding complexities reported in [BBDV20]. Note that in this table, we have recalculated all computational complexities in terms of number of encryptions.

The rest of the paper is organized in the following manner. In Section 2, we begin by presenting a mathematical description of LowMC and some information about the LowMC cryptanalysis challenge. In Section 3, we list out some of the issues with the computational complexity reported in [BBDV20] and explain how we have tried to compute all complexities in terms of number of encryptions. In Section 4, we present our attack on the 2-round and 3-round LowMC instances with complete non-linear layers. In Section 5, we present our attack on the $0.8 \cdot \lfloor \frac{n}{s} \rfloor$ and $\lfloor \frac{n}{s} \rfloor$-LowMC instance with partial non-linear layers. In Section 6, we present some experimental results on reduced LowMC instances with smaller blocksizes. This is done to prove that the attacks presented in Section 4, 5 can indeed be applied to full-size LowMC instances. Section 7 concludes the paper.

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. 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. 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

Table 1: Summary of results. Note for the complexity is given in #Encryptions

Instance	n	s	r	Type of Attack	Recalculated Complexity	Reference
Full S-box layer	129	43	2	Linearization	2^{91}	[BBDV20]*
	192	64			2^{134}	
	255	85			2^{176}	
Partial S-box layer	128	1	$0.8 \times \lfloor \frac{n}{s} \rfloor$	Linearization	2^{102}	[BBDV20]*
	192	1			2^{153}	
	256	1			2^{204}	
Partial S-box layer	128	10	$0.8 \times \lfloor \frac{n}{s} \rfloor$	Linearization	2^{103}	[BBDV20]*
	192	10			2^{163}	
	256	10			2^{203}	
Full S-box layer	129	43	2	Equation solving	2^{102}	[Din]**
			3		2^{108}	
			4		2^{113}	
Full S-box layer	192	64	2	Equation solving	2^{153}	[Din]**
			3		2^{162}	
			4		2^{170}	
			5		2^{175}	
Full S-box layer	255	85	2	Equation solving	2^{204}	[Din]**
			3		2^{216}	
			4		2^{226}	
			5		2^{232}	
Full S-box layer	129	43	2	2-Stage MITM	$\mathbf{2^{81}}$	Sec 4
	192	64			$\mathbf{2^{122}}$	
	255	85			$\mathbf{2^{164}}$	
Full S-box layer	129	43	3	2-Stage MITM	2^{123}	Sec 4
	192	64			2^{186}	
	255	85			2^{248}	
Partial S-box layer	128	1	$0.8 \times \lfloor \frac{n}{s} \rfloor$	2-Stage MITM	$\mathbf{2^{101}}$	Sec 5
	192	1			$\mathbf{2^{151}}$	
	256	1			$\mathbf{2^{202}}$	
Partial S-box layer	128	1	$\lfloor \frac{n}{s} \rfloor$	2-Stage MITM	$\mathbf{2^{125}}$	Sec 5
	192	1			$\mathbf{2^{189}}$	
	256	1			$\mathbf{2^{253}}$	
Partial S-box layer	128	10	$0.8 \times \lfloor \frac{n}{s} \rfloor$	2-Stage MITM	$\mathbf{2^{91}}$	Sec 5
	192	10			$\mathbf{2^{149}}$	
	256	10			$\mathbf{2^{188}}$	
Partial S-box layer	128	10	$\lfloor \frac{n}{s} \rfloor$	2-Stage MITM	$\mathbf{2^{111}}$	Sec 5
	192	10			$\mathbf{2^{179}}$	
	256	10			$\mathbf{2^{238}}$	

*Complexities recalculated and do not always match those reported in [BBDV20]

**[Din] reports complexities in bit operations. We recalculate them in number of encryptions.

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. the data complexity $d = 1$.

- **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 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 $1.2 \times \lfloor \frac{n}{s} \rfloor$. 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.

3 Linearization Attack

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 1. *[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 the LowMC S-box output bits can be written as:

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

The same is true for the inverse LowMC S-box (which is incidentally affine equivalent to the forward S-box):

$$\begin{aligned} t_0 &= x_0 + x_1 + x_1 \cdot x_2 && = f \cdot (x_1 + x_2 + 1) + x_0 + x_1, \\ t_1 &= x_1 + x_0 \cdot x_2 && = f \cdot (x_0 + x_2 + 1) + x_1, \\ t_2 &= x_0 + x_1 + x_2 + x_0 \cdot x_1 && = f \cdot (x_0 + x_1 + 1) + x_0 + x_1 + x_2. \end{aligned}$$

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. However the authors pointed out 2 issues in this approach:

1. If the total number of S-boxes in the encryption circuit is t, then the algorithm requires in the worst case requires at least 2^t computations of the encryption function (for the verification of each computed candidate key). It additionally requires 2^t Gaussian elimination calculations. For large blocksizes, the authors claimed this could prove to be a significant bottleneck.
2. 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.

The authors could not find a closed form for the value of d_A and so could not assign a tight bound on the computational complexity incurred in this approach. However we find that some of these issues can be resolved to get a closed form expression of the complexity of the linearization algorithm. First of all, the expected number of solutions for the system $A \cdot [k_0, k_1, \ldots, k_{n-1}]^T = const$ is 1 if the system is random. If *const* lies in the image of the linear transformation defined by A then the system has 2^{d_A} solutions, and it has 0 solutions otherwise. Now the probability that *const* lies in the image of A is exactly 2^{-d_A} and so the average number of solutions by Bayes theorem is $2^{d_A} \cdot 2^{-d_A} + (1 - 2^{-d_A}) \cdot 0 = 1$, and testing this solution costs us one encryption.

Note that multiplying an $n \times n$ matrix with an n-bit column vector requires n^2 bit operations. Every LowMC round therefore requires at least $2n^2$ bit operations (n^2 for computing the affine layer and another n^2 for generating the round key). Assuming calculation of the S-box layer can be done in linear time using a lookup table and also since key xor with state also takes linear time, the sum total of all the other bit operations in the round are linear in n. Suppressing these, the total bit operations required in performing a LowMC encryption is around $2rn^2$. Solving a system of linear equations by Gaussian elimination (GE) costs around n^3 bit operations which is equivalent to $\frac{n^3}{2rn^2} = \frac{n}{2r}$ encryptions.

Also note the computational complexity required to formulate the linear system $A \cdot [k_0, k_1, \ldots, k_{n-1}]^T = const$. We argue that this is equivalent to n encryptions. After guessing the majority bits, the system becomes completely linear. Therefore finding the i-th column of A and the i-th bit of $const$ is equivalent to performing one encryption with the basis key vector $[0, 0, \ldots, k_i, \ldots, 0, 0]$. Hence the result follows. Therefore the total computational complexity required to perform the attack using only linearization in terms of number of encryptions is

$$2^{rs} \text{ (Guessing majority bits)} \times [\ n \text{ (Formulating the linear system)} + \frac{n}{2r} \text{ (Solving the linear system)} + 1 \text{ (Testing one solution on average)}].$$

We can simplify this to $n \cdot 2^{rs}$ encryptions. Also note that 2^{rs} is the worst case complexity for guessing rs bits. The average case complexity is 2^{rs-1}. However since we want to compare this complexity to the complexity of exhaustive search 2^n which is also a worst case complexity we use 2^{rs} for all our complexity estimations.

3.1 Improving complexity using Gray-Code based approach

The above complexity can be significantly improved if one were to make the majority guesses in a Gray-code like manner. Recall that the encoding is defined as follows: $\textsf{Graycode}(i) = i \oplus (i \gg 1)$. Note that hamming difference between $\textsf{Graycode}(i)$ and $\textsf{Graycode}(i+1)$ is always 1 for all values of i. The idea is instead of ordering the majority guesses in lexicographic order, we use the order defined by the Gray-code, i.e. in the i-th step the majority guess sequence is the binary string defined by the bits of $\textsf{Graycode}(i)$. When this is done the matrix A defined above, changes very little from iteration i to $i+1$. Thus having already constructed A in the i-th iteration, the corresponding construction in the $i+1$-th iteration can be done much faster and so the cost of formulating the linear system of equations defined by Eqn (1) can be amortized over all the majority guesses.

Let us state the algorithm formally. Let $M = m_0, m_1, \ldots, m_{s-1}, \quad m_s, m_{s+1}, \ldots, m_{2s-1}, \ldots, m_{(r-1)s}, m_{(r-1)s+1}, \ldots, m_{rs-1}$ be the rs majority guesses for the s number of S-boxes in each of the r rounds. Let M_i denote the value of the

string M at the i-th iteration which we want to be equal to $\textsf{Graycode}(i)$. Let the linearized system of equations at the i-th iteration be denoted as $A_i \cdot k = c_i$. We want to determine how A_{i+1}, c_{i+1} relate with respect to A_i, c_i. Let $x \to Tx \oplus v$ be the linear map from $\{0,1\}^n \to \{0,1\}^n$ that is obtained as a result of linearizing the S-boxes in any single round with the majority value string Str (note that T is an $n \times n$ matrix and v is a n-element vector). Let $x \to T'x + v'$ be the corresponding map when the majority string is $Str \oplus \mathbf{e}_t$ (here $\mathbf{e}_t$ denotes the t-th unit vector of length s and $0 \leq t < s$). Then we define $\Delta_t = T \oplus T'$ and $\lambda_t = v \oplus v'$, so that $\Delta_t x + \lambda_t$ denotes the change of linear map when the majority guess changes at the t-th S-box.

Let L_a denote the $n \times n$ matrix used in the linear layer in the a-th round (with $1 \leq a \leq r$). Also, let $\textsf{Graycode}(i) \oplus \textsf{Graycode}(i+1) = \mathbf{e}_j$ for some j (by slight abuse of notation $\mathbf{e}_j$ here denotes the j-th unit vector of length rs). If $j < s$, then it can be deduced that $A_i \oplus A_{i+1} = (\prod_{a=1}^{r} L_a) \cdot \Delta_j := B_j$ (say) and $c_i \oplus c_{i+1} = (\prod_{a=1}^{r} L_a) \cdot \lambda_j := b_j$. If $j \in [(u-1)s, us-1]$, which means that the change of majority guess occurs in the u-th round, then denote $j' = j - (u-1)s$. B_j is now defined as $A_i \oplus A_{i+1} = (\prod_{a=u}^{r} L_a) \cdot \Delta_{j'}$ and $b_j = (\prod_{a=u}^{r} L_a) \cdot \lambda_{j'}$. Note that it is thus possible to precompute for all $j \in [0, rs-1]$ the matrix-vector pair (B_j, b_j) before the linearization step begins. Thus the linearization attack can be restated as follows:

1. For all $j \in [0, rs-1]$ precompute the matrix-vector pair (B_j, b_j).
2. Compute A_0, c_0 and try to solve the system $A_0 \cdot k = c_0$ using GE.
3. For $i = 1 \to 2^n - 1$ do
   - The majority guess is $M_i = \textsf{Graycode}(i)$.
   - Let $\textsf{Graycode}(i) \oplus \textsf{Graycode}(i-1) = \mathbf{e}_j$.
   - Calculate $A_i = A_{i-1} \oplus B_j$ and $c_i = c_{i-1} \oplus b_j$.
   - Try to solve the system $A_i \cdot k = c_i$ using GE.

Note that since none of the B_j's are sparse matrices, we can not devise a quicker method of doing GE on A_i from the knowledge of steps involved in the GE of A_{i-1}. The additional complexity of constructing A_i, c_i at each step is given by a matrix and vector addition and so equal to $n^2 + n$ bit operations which roughly corresponds to $\frac{n^2+n}{2rn^2} \approx \frac{1}{2r}$ encryption operations. Thus if P denotes the cost involved in pre-computation (which is at most a polynomial in rs) then the total complexity of the method can be written as $P + 2^{rs} \cdot (\frac{n}{2r} + 1 + \frac{1}{2r}) \approx \frac{n}{2r} \cdot 2^{rs}$ encryptions which gives us an improvement of a factor of $2r$ over the naive linearization method of the previous subsection. We have recalculated the complexities in Table 1 using this expression.

4 Attacking instances with complete S-box Layers

4.1 MITM attack on 2-round LowMC in [BBDV20]

Before we present our attack, let us summarize the attack in [BBDV20] for better understanding of the process. The attack is summarized in Fig. 2. The

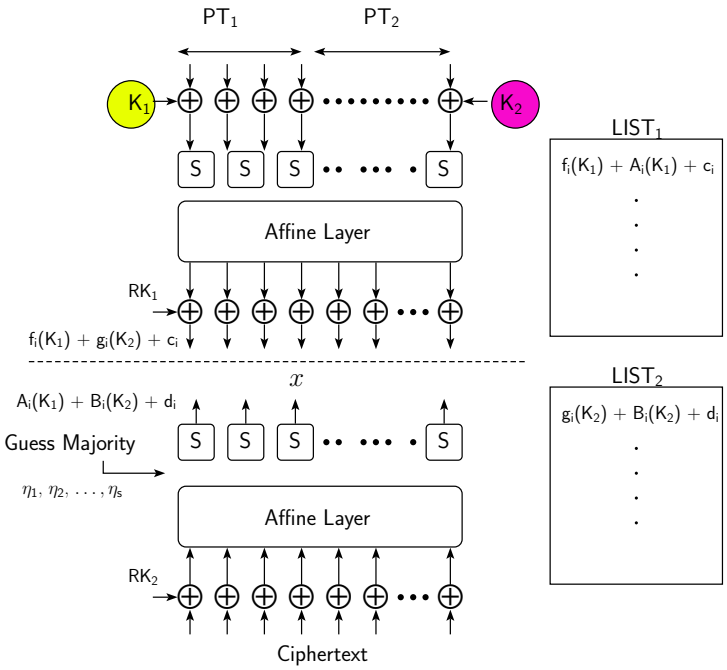


Fig. 2: Meet in the Middle attack in [BBDV20]

idea is as follows: let us denote $K = [k_0, k_1, \ldots, k_{n-1}]$ to be the whitening key or the master key. Let us 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. We denote by R_1, R_2 the first and second round functions i.e. $R_1(pt + K, RK_1) = x$ 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 whitening key K.

The idea is to formulate equations for the bits of x from both the plaintext and ciphertext side. Let us begin from the plaintext in the forward direction. Note that K_1 and K_2 have to be chosen so that the bits of K_1 and K_2 are never multiplied in the first round function. For example if the number of S-boxes in each round $s = n/3$ is odd, then t can be chosen to be $3(s-1)/2$ (else $t = 3s/2$). This way, K_1 and K_2 both contain close to $n/2$ key bits: the bits of K_1 after whitening are input to the first $(s-1)/2$ S-boxes and K_2 to the remaining $(s+1)/2$ S-boxes if s is odd (else both are input to $s/2$ S-boxes each). 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.

Now let us consider the expression for x from the ciphertext side in the backward direction. 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 each x_i can be further written as $x_i = A_i(K_1) + B_i(K_2) + d_i$, $\forall\ i \in [0, n-1]$, where each $A_i,\ B_i$ are linear functions over K_1, K_2 and d_i is a single bit constant. 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:

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

Thereafter the attack is straightforward: first the algebraic expressions of f_i, g_i and c_i for all $i \in [0, n-1]$ are calculated. Then for each of the 2^s guesses of the second round majority values:

1. A hash table LIST_1 indexed by the n-bit vector $[f_i(K_1) \oplus A_i(K_1) \oplus c_i],\ \forall\ i \in [0, n-1]$ is created (2^t evaluations). Note that each evaluation is done for only t of the n key variables and costs roughly $\frac{t}{n}$ of a round computation. Hence the computational complexity incurred in this step is $\frac{t}{4n} \cdot 2^t$ encryptions. Let us argue this point more closely. Note that in the above expression the $f_i(K_1)$ terms are always constant and does not change with every new guess of majority values. Thus we do not have to re-calculate it every new majority guess, and so this expression ($\forall\ i \in [0, n-1]$) can be calculated once and stored in a table. The part that varies with every new majority guess is $A_i(K_1) \oplus c_i$: note that calculating the n bit-values $A_i(K_1)$ is equivalent to a matrix-vector multiplication between a $n \times t$ matrix and the t-element vector K_1 and thus takes around nt bit operations. Adding c_i and the precomputed $f_i(K_1)$ requires $2n$ more bit operations and so a total of $nt+2n$ bit operations are required at every step. Since $2rn^2 = 4n^2$ bit operations are required in a single 2-round LowMC encryption, this corresponds to $\frac{nt+2n}{4n^2} \approx \frac{t}{4n}$ encryptions, and so the result follows.
2. A hash table LIST_2 indexed by the n-bit vector $[g_i(K_2) \oplus B_i(K_2) \oplus d_i],\ \forall\ i \in [0, n-1]$ is created (2^{n-t} evaluations). By following the previous logic this is computationally equivalent to $\frac{n-t}{4n} \cdot 2^{n-t}$ encryptions.

As a final remark, note that the complexity required to formulate the expressions $x_i = A_i(K_1)+B_i(K_2)+d_i$ and hence $f_i(K_1)+A_i(K_1)+c_i$ and $g_i(K_2)+B_i(K_2)+d_i$ is around $O(n)$ encryptions as explained in the previous section. However, this only appears as an additive term along with $\frac{t}{4n} \cdot 2^t$ and $\frac{n-t}{4n} \cdot 2^{n-t}$ and since it is much less as compared to both these terms, it can be ignored for simplicity. Although, it was not mentioned in [BBDV20], a Gray-code like approach as outlined in Sec 3.1 may be adopted here too, but since the cost of formulating the linear system is not the dominant term in the final complexity estimate, it does not reduce the computational cost significantly.

A collision in the 2 lists gives us a candidate key and there are on average $2^t \cdot 2^{n-t} \cdot 2^{-n} = 1$ collisions in every MITM stage. Then a check is performed to see if the majority bits calculated for this candidate key are consistent with the initial guess $\eta_1, \ldots, \eta_s$. If yes, the attack terminates. The total complexity for the steps inside the iterations is given as $T = 2^s \cdot (\frac{t}{4n} \cdot 2^t + \frac{n-t}{4n} \cdot 2^{n-t})$ encryptions.

The cost for precomputing the values $f_i(K_1)$ and $g_i(K_2)$ over all the points in their input spaces can be done by using Möbius transforms over the respective algebraic forms. Since any t variable Boolean function can be evaluated in $t \cdot 2^{t-1}$ bit operations using this method, the total complexity of evaluating them is around $n \cdot (\frac{t}{2} \cdot 2^t + \frac{(n-t)}{2} \cdot 2^{n-t})$ bit-operations. This is considerably lower than the complexity T of the MITM part. Specifically, for $n = 129$, we can take $t = 63$ and $n - t = 66$. The total complexity of the attack is around $2^{43} \cdot (2^{60} + 2^{63}) \approx 2^{43+63} = 2^{106}$ encryptions.

4.2 2-stage MITM attack on 2-rounds with full S-box layer

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

$$f_i(K_1) + A_i(K_1) + c_i = g_i(K_2) + B_i(K_2) + d_i, \ \forall \ i \in [0, n-1]. \tag{2}$$

Note that the functions $A_i, \ B_i$ are linear and $f_i, \ g_i$ are quadratic. It can be seen that for Equation (2) to hold we need not split K in such a way that K_1 and K_2 have approximately $n/2$ bits. We can, for example, also split K so that K_1 has around $n/3$ and K_2 has around $2n/3$ bits. The only condition that must be satisfied is that the sizes of K_1 and K_2 are chosen so that they are never mixed multiplicatively in the first round. It is easy to see that if we choose $t = |K_1|$ and $n - t = |K_2|$ to be multiples of 3 then this condition is automatically satisfied.

Note that, $f_i, \ g_i$ can be expressed as affine functions 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

$$\overline{f}_i(\overline{K}_1) + A_i(\overline{K}_1) + c_i + d_i = g_i(K_2) + B_i(K_2), \ \forall \ i \in [0, n-1], \tag{3}$$

where if $K_1 = \ [k_0, k_1, k_2, \ldots, k_{3w-3}, k_{3w-2}, k_{3w-1}]$, we define

$$\overline{K}_1 = [k_0, k_1, k_2, k_0k_1, k_1k_2, k_2k_0, \ldots\ldots\ldots, k_{3w-3}, k_{3w-2}, k_{3w-1}, \\ k_{3w-3}k_{3w-2}, k_{3w-2}k_{3w-1}, k_{3w-1}k_{3w-3}].$$

Since K_1 only has the first $t = 3w$ bits of the master key and so $\overline{K}_1$ is of size $6w$. 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 $n/3$ key bits i.e. $w \approx n/9$ and hence K_2 contains the remaining $2n/3$ key bits. Since ϕ is seen as a linear code, let

$\mathbf{G}$ be the corresponding generator matrix (of size $n \times 6w \approx n \times 2n/3$), which can be efficiently constructed from the algebraic forms of the functions F_i. Let $\mathbf{H}$ be the parity check matrix of the code (of size $(n-6w) \times n \approx n/3 \times n$). 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 (3), 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). This follows from the fact that since $\mathbf{G}$ and $\mathbf{H}$ are the generator and parity check matrices of a linear code, we must have $\mathbf{H} \cdot \mathbf{G} = 0$.

We can split K_2 into two halves K_{21} and K_{22} such that both halves contain approximately $n/3$ key bits each. Let's say $|K_{21}| = 3u$ and $|K_{22}| = n-3w-3u$ (our strategy would be to have $3u \approx n-3w-3u$ so that the halves are of equal size). 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 if we take $|K_{21}|$ and $|K_{22}|$ to be multiples of 3, so that the bits of K_{21} and K_{22} after xor with the plaintext are input to different S-boxes. Due to the structure of LowMC, the quadratic terms from adjacent S-boxes do not combine multiplicatively after one round and so the separation into the 2 expressions is possible. Define the n-bit vectors:

$$M_1 = [g_0^1(K_{21}) + B_0^1(K_{21}), \ldots, g_{n-1}^1(K_{21}) + B_{n-1}^1(K_{21})]^T, \text{ and}$$
$$M_2 = [g_0^2(K_{22}) + B_0^2(K_{22}), \ldots, g_{n-1}^2(K_{22}) + B_{n-1}^2(K_{22})]^T.$$

Note that if Eqn (3) for $i = 0, 1, \ldots, n-1$, can be written together as a vector equation. The right hand side of the vector equation is essentially $M_1 + M_2$. We have already seen that the left hand side of the vector equation when multiplied by $\mathbf{H}$ results in the vector $\mathbf{H} \cdot Con = e$. Multiplying the right side of the vector equation by $\mathbf{H}$, we get the matrix equation:

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

Pre-computation: In this phase we try and compute some expressions that remain constant over different majority guesses. We compute the following vectorial functions over all points over its input space: **(a)** $f_i(K_1)$, $\forall i \in [0, n-1]$ over input space of K_1 i.e $\{0,1\}^{3w}$, **(b)** $g_i^1(K_{21})$, $\forall i \in [0, n-1]$ over input space of K_{21} i.e $\{0,1\}^{3u}$ and **(c)** $g_i^2(K_{22})$, $\forall i \in [0, n-1]$ over input space of K_{22} i.e $\{0,1\}^{n-3u-3w}$. Using Möbius transform the number of bit-operations required are

$$n \cdot \left(\frac{3w}{2} \cdot 2^{3w} + \frac{3u}{2} \cdot 2^{3u} + \frac{n-3u-3w}{2} \cdot 2^{n-3u-3w} \right).$$

This follows since any t-variable Boolean polynomial can be evaluated over all its input space using Möbius transform using $t \cdot 2^{t-1}$ bit operations.

1st MITM stage: Note that M_1 and M_2 only contain expressions on the key bits 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 $(n-6w)$, n-bit vector pairs $\mathbf{H}\cdot M_1,\ M_1$ for all 2^{3u} values of K_{21}. And similarly the list L_2 contains the $(n-6w), n$-bit vector pairs $\mathbf{H}\cdot M_2+e, M_2$ for all $2^{n-3w-3u}$ values of K_{22}. We look for a collision in the $n-6w$ co-ordinates of these lists. We are expected to get around $2^{3u+(n-3w-3u)-(n-6w)} \approx 2^{3w}$ collisions. Thus in the process we get 2^{3w} key values for the key bit set $K_2 = (K_{21}, K_{22})$. For computing each entry in the list L_1 we do the following:

1. Compute the vectorial linear functions $B_0^1, B_1^1, \ldots, B_{n-1}^1$ over a given point k in K_{21}. Each such computation takes $|K_{21}|\cdot n = 3un$ bit operations.
2. Add to the corresponding precomputed vector $g_i^1(k),\ \forall i \in [0, n-1]$. This requires n bit operations.
3. Multiply by $\mathbf{H}$. Each such computation takes $(n-6w)\cdot n$ bit operations.

This is computationally equivalent to $\frac{3un+n+(n-6w)n}{2rn^2} \approx \frac{3u+n-6w}{4n}$ of an encryption for $r=2$. A similar argument holds for L_2. Hence the total computational cost incurred in this step is $\frac{3u+n-6w}{4n}\cdot 2^{3u} + \frac{2n-9w-3u}{4n}\cdot 2^{n-3w-3u}$ encryptions.

2nd MITM: Let us now turn to Eqn (2). The left side of this equation is defined over approximately the $3w$-bit set K_1 which can have 2^{3w} values in total. And we have just reduced K_2 to a set of 2^{3w} values. Thus the next MITM is making two more lists $L_3,\ L_4$ of size 2^{3w} each in the following way. L_3 contains all 2^{3w} n-bit vectors $[f_i(K_1)\oplus A_i(K_1)\oplus c_i \oplus d_i],\ \forall\, i \in [0, n-1]$ enumerated for all the 2^{3w} values of K_1. For all the 2^{3w} values of K_2 that have passed the previous MITM step we make the list L_4 containing the n-bit vector $[g_i(K_2)\oplus B_i(K_2)],\ \forall\, i \in [0, n-1]$. We now look for a collision between L_3 and L_4. On average we have $2^{3w+3w-n} = 2^{6w-n} < 1$ collisions. This means that the correct key K will necessarily by the output of one of these MITM steps for the correct guess of majority bits in the second round. For constructing L_3 we need to compute the n linear functions $A_i(K_1)$ over the $3w$-bit variable K_1 which by the previous logic, requires $3wn$ bit operations each and then n bit operations for addition to the precomputed vector $f_i(K_1)$. Populating L_4 requires computing $[g_i(K_2)\oplus B_i(K_2)]$ for all the K_2 that have passed the previous MITM step. However we can compute this vector by simply adding the $M_1,\ M_2$ vectors that have collided in the previous MITM stage. This stage therefore requires $\frac{3wn+n}{4n^2}\cdot 2^{3w} + \frac{n}{4n^2}\cdot 2^{3w} \approx \frac{3w}{4n}\cdot 2^{3w}$ encryptions. 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. Pre-compute $f_i(K_1),\ g_i^1(K_{21}),\ g_i^2(K_{22}), \forall i \in [0, n-1]$ over their respective input spaces.
3. Guess the majority values $\eta_1, \ldots, \eta_s$ at the output of 2nd round S-box layer as in the previous attack. 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 $(n - 6w)$-bit vector $\mathbf{H} \cdot M_1$.
- For all possible values of K_{22}, create a hash table L_2 indexed by the $(n - 6w)$-bit vector $\mathbf{H} \cdot M_2 + e$.
- 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 $[f_i(K_1) \oplus A_i(K_1) \oplus c_i \oplus d_i],\ \forall\ i \in [0, n-1]$.
- For all values of $K_2 \in L$, create a hash table L_4 indexed by the n-bit vector $[g_i(K_2) \oplus B_i(K_2)],\ \forall\ i \in [0, n-1]$.
- 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: We first consider the time complexity. For each guess of $2^s = 2^{n/3}$ majority values, we have to perform a Gaussian elimination and 2 MITM steps. The cost of Gaussian elimination and the linear terms required to formulate A_i, B_i, d_i and pre-computation may be ignored in comparison with $2^{n/3}$. Hence the total time complexity for this attack is around

$$2^{n/3} \cdot \left(\frac{3u + n - 6w}{4n} \cdot 2^{3u} + \frac{2n - 9w - 3u}{4n} \cdot 2^{n-3w-3u} + \frac{3w}{4n} \cdot 2^{3w} \right). \quad (4)$$

For $w = u = n/9$, the above evaluates to $2^{n/3} \cdot ((\frac{1}{6} + \frac{1}{6} + \frac{1}{12}) \cdot 2^{n/3}) = \frac{5}{12} \cdot 2^{2n/3} \approx 2^{2n/3-1.26}$ encryptions.

Memory Complexity: In the first MITM stage, we created 2 lists L_1, L_2 which contain $(n-6w)$, n-bit vector pairs for 2^{3u} possible values of K_{21} and $(n-6w)$, n-bit vector pairs for $2^{n-3w-3u}$ possible values of K_{22}, respectively. Note that in practice, 2 different lists are not necessary. We can instead insert each new vector of L_1 and L_2 into a single hash table. The memory complexity here is $(2n-6w) \cdot (2^{3u} + 2^{n-3w-3u})$ bits. In the second MITM stage, we create 2 more lists L_3, L_4, both containing 2^{3w} n-bit vectors. By similar logic, memory complexity here is thereby $2n \cdot 2^{3w}$ bits. The pre-computation part generates n-bit vectors over the input spaces of K_1, K_{21}, K_{22}. Hence the memory complexity here is $n \cdot (2^{3w} + 2^{3u} + 2^{n-3u-3w})$ bits. The total memory complexity for this attack is around

$$(2n - 6w) \cdot (2^{3u} + 2^{n-3w-3u}) + 2n \cdot 2^{3w} + n \cdot (2^{3w} + 2^{3u} + 2^{n-3u-3w}) \text{ bits.} \quad (5)$$

If we look at concrete parameters, for $n = 129$ and $s = 43$, we can choose the parameters in the following manner: we can choose $w = u = 14$, which makes $|K_1| = 42$ and $|K_2| = 87$ and hence $|K_{21}| = 42$ and $|K_{22}| = 45$. The parity check matrix $\mathbf{H}$ is of size $(n - 6w) \times n = 45 \times 129$, which makes $\mathbf{H} \cdot M_1$ and $\mathbf{H} \cdot M_2 + e$ both 45-bit vectors. After the first MITM stage the number of remaining candidates for K_2 is $\approx 2^{|K_{21}|+|K_{22}|-45} = 2^{42}$. The complexity of the first MITM

stage is thus $\frac{1}{6} \cdot (2^{45} + 2^{42}) \approx \frac{1}{6} \cdot 2^{45} \approx 2^{42.4}$ encryptions. The second MITM stage requires $\frac{1}{12} \cdot 2^{42} = 2^{38.4}$ encryptions. Hence the total attack complexity is $2^s \cdot (2^{42.4} + 2^{38.4}) \approx 2^{85}$ encryptions and around 2^{53} bits of memory. This is lower than the linearization attack by a factor of 2^6 for this LowMC instance.

4.3 Extending attack to 3-rounds

The attack can be extended to 3-round LowMC in which we keep the basic character of the algorithm and run it by guessing the majority values of the last 2 rounds and linearizing both of them simultaneously. Hence a total of 2^{2s} values would need to be guessed in stead of 2^s. All other steps remain the same. Thus the computational complexity will be given by:

$$2^{2n/3} \cdot \left(\frac{3u + n - 6w}{6n} \cdot 2^{3u} + \frac{2n - 9w - 3u}{6n} \cdot 2^{n-3w-3u} + \frac{3w}{6n} \cdot 2^{3w} \right).$$

This is so since encryption is now given by $2rn^2 = 6n^2$ bit operations. The memory complexity is essentially the same as in the 2-round attack. For $w = u = n/9$, the above evaluate of computational complexity is $2^{2n/3} \cdot ((\frac{1}{9} + \frac{1}{9} + \frac{1}{18}) \cdot 2^{n/3}) \approx \frac{5}{18} \cdot 2^n$ encryptions, which is better than exhaustive search by a factor equal to approximately 2 bits. For $n = 129$ and $s = 43$, using the values $w = 14$, $|K_1| = 42$, $|K_{21}| = 42$ and $|K_{22}| = 45$, we get $\frac{1}{9} \cdot (2^{45} + 2^{42}) \approx 2^{41.8}$ encryptions for the first MITM. The second MITM requires $\frac{1}{18} \cdot 2^{42} \approx 2^{37.8}$ encryptions. The total complexity is therefore $2^{2s} \cdot (2^{41.8} + 2^{37.8}) \approx 2^{128}$ encryptions.

4.4 Speedup using Gray-Codes

There are 3 places in the above process where a speed-up may be applied using a Gray-code like approach.

1. By ordering the majority guesses in a Gray-code like manner as in Sec 3.1 so that the affine expressions formed after linearizing the S-boxes can be generated more efficiently. But we have already seen that this does not result in significant speed-up when employed along with MITM.
2. Using a Gray-code like approach to do the pre-computations.
3. Using a Gray-code like approach to generate the values of the expressions that are inserted in the tables in each of the MITM stages. We will see how optimizing this stage results in significant speed-up.

There are several methods of evaluating an n-variable Boolean function over all the 2^n points of its input space, given its algebraic expression. One such method, as we have already seen is the Möbius transform which evaluates the function in-place by performing around $n \cdot 2^{n-1}$ bit operations. However the method we will use for this method is the Gray-code based approach suggested by [BCC+10] which finds all roots of a polynomial over GF(2) by evaluating it over all points of its input space by traversing the space in a Gray-code like manner. We start with the following theorem from [BCC+10].

Theorem 1. *[BCC+10] All the zeroes of a single multivariate polynomial f in n variables of degree d can be found in essentially $d \cdot 2^n$ bit operations (plus a negligible overhead), using n^{d-1} bits of read-write memory, and accessing n^d bits of constants, after an initialization phase of negligible complexity $O(n^{2d})$.*

We present a top-level overview of the approach used in this paper. Consider the derivative $\frac{\delta f}{\delta i} : \mathbf{x} \rightarrow f(\mathbf{x} + \mathbf{e}_i) \oplus f(\mathbf{x})$. Then for any vector $\mathbf{x}$, we have $f(\mathbf{x} + \mathbf{e}_i) = f(\mathbf{x}) \oplus \frac{\delta f}{\delta i}(\mathbf{x})$. If the algebraic degree of f is d then $\frac{\delta f}{\delta i}$ is of degree $d-1$. Thus the idea is to calculate $\frac{\delta f}{\delta i}$ recursively for lower degrees till at the lowest level of recursion $\frac{\delta f}{\delta i}$ is a constant. Since we will only use this method to evaluate linear or quadratic functions, we will use the method outlined in [BCC+10, pg 209, Fig. (b)], that specifically caters to the case when f is of degree less than or equal to 2. When we use this approach to optimize the pre-computation part, we can evaluate each t-variable quadratic Boolean function in 2^{t+1} bit-operations. As a result the pre-computation cost can be brought down to $2n \cdot \left(2^{3w} + 2^{3u} + 2^{n-3u-3w}\right)$ bit-operations. However, note that the pre-computation is not the most dominant term in the total computational cost, and so this gives only a slight improvement.

We now see how we can improve the complexity of the MITM stages by using this approach. Note that we only evaluate linear functions inside the iterations for each majority guess. Since only 2^t bit-operations are required to evaluate any linear function using the Gray-code approach we can accelerate this part considerably. Note that in L_1 we need to store both $\mathbf{H} \cdot M_1$ and M_1. To do this, we begin by computing the quadratic expressions each one of the n bits M_1 and then each of the $(n-6w)$-bits given by $\mathbf{H} \cdot M_1$. We use the Gray-code approach of [BCC+10], to evaluate these functions over all the points of their input domains. The number of bit operations required are therefore $n \cdot 2^{3u+1} + (n-6w) \cdot 2^{3u+1} \approx \frac{2n-6w}{2rn^2} \cdot 2^{3u+1}$ encryptions. Similarly the list L_2 would require around $\frac{2n-6w}{2rn^2} \cdot 2^{n-3u-3w+1}$ encryptions.

The lists L_3, L_4 are simpler to construct. For L_3 we need to compute the n linear functions $A_i(K_1)$ which requires $n \cdot 2^{3w}$ bit operations each and then add to the precomputed vector $f_i(K_1)$. Populating L_4, as before can be done by simply adding the M_1, M_2 vectors that have collided in the previous MITM stage. This stage therefore requires $\frac{2n}{2rn^2} \cdot 2^{3w} + \frac{n}{2rn^2} \cdot 2^{3w} \approx \frac{3n}{2rn^2} \cdot 2^{3w}$ encryptions. This reduces the main terms of the computational complexity to

$$T = 2^{\frac{(r-1)n}{3}} \cdot \left(\frac{n-3w}{rn^2} \cdot 2^{3u+1} + \frac{n-3w}{rn^2} \cdot 2^{n-3u-3w+1} + \frac{3n}{2rn^2} \cdot 2^{3w} \right) \text{ encryptions}$$

For $n = 129$, $r = 2$ and $u = w = 14$, we have $T = 2^{80.7}$ encryptions. For $n = 129$, $r = 3$ and $u = w = 14$, we have $T = 2^{123.2}$ encryptions. The memory complexity of this attack is the same as the attack in the previous sub-section plus the additional cost for storing tables required for fast Gray-code based evaluations. Using Theorem 1, this additional memory is $(3u)^2 \cdot (2n-6w) + (n-3u-3w)^2 \cdot (2n-6w) + (3w) \cdot n$ bits which is negligible when compared to the space occupied by the lists.

5 2-Stage MITM attack on partial S-box layers

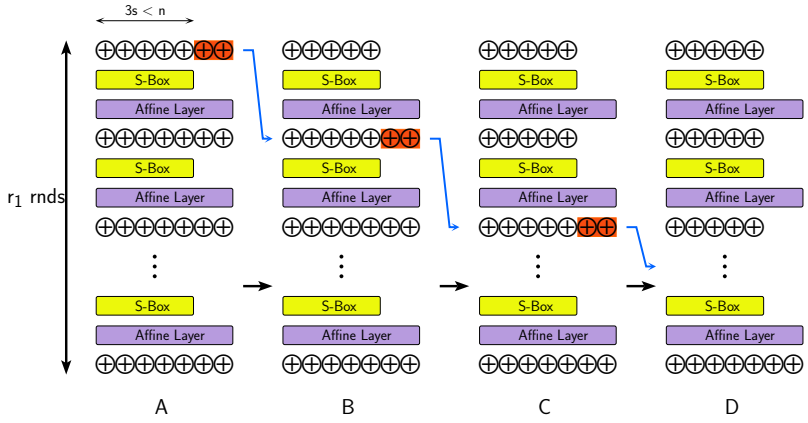


Fig. 3: Transforming the round function in the first r_1 rounds. From A $\rightarrow$ B, the key material not added to bits input to the S-box in round 1 (shown in orange background) are carried to the next round, through the affine layer and merged with the round key in round 2. B $\rightarrow$ C $\rightarrow$ D do the same from the second round onwards. Figure taken from [BBDV20]

In order to perform a MITM on the partial S-box layer instances of LowMC, we use a trick used in both [BBDV20,RST18] to transform some of the initial and final rounds so that the total number of different key bits involved in these rounds is $3s$ per round. The transformations are shown in Figs. 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 L is a linear function, we 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. To mount this attack let us split the LowMC into 4 parts as shown in Fig. 5:

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

Let the set of round key bits in the first a, b and the last c rounds be denoted as

$$K_a = [\kappa_0, \kappa_1, \ldots, \kappa_{3sa-1}],\ K_b = [\kappa_{3sa}, \kappa_{3sa+1}, \ldots, \kappa_{3sa+3sb-1}],\ \text{and}$$
$$K_c = [\kappa_{n-3sc}, \kappa_{n-3sc+1}, \ldots, \kappa_{n-1}].$$

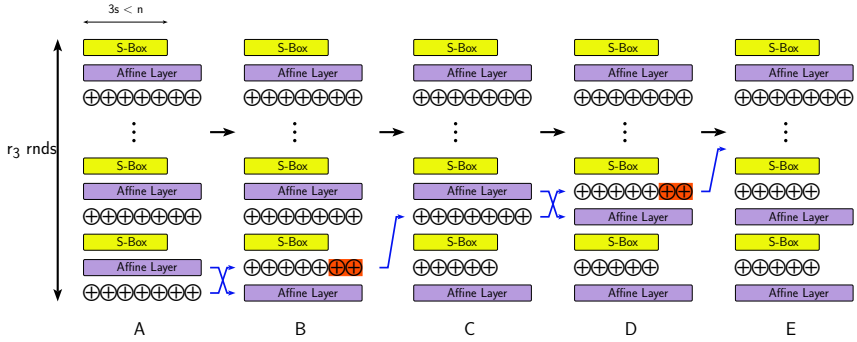


Fig. 4: Transforming the round function in the final r_3 rounds. A $\rightarrow$ B flips the order of the last round Affine layer and round key xor. B $\rightarrow$ C takes the bits of the last round key that are not added to S-box outputs (shown in orange background), and brings them back by 1 round 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. Figure taken from [BBDV20]

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. Note that we implicitly assume here that $n \geq 3s(a+b+c)$.

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. 5. Observe the middle b and $d = r - a - b - c$ rounds closely, as seen in Fig. 6. 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 first aim is to find a linear expression relating the x_i's, y_i's and z_i's and the key bits. Let $D = [D_0, D_1, \ldots, D_{n-1}]$ be the output of the first of the b rounds (see Fig. 6). Then we can write $D = \mathbf{Lin}_1(z_0, z_1, \ldots, z_{3s-1}, x_{3s}, x_{3s+1}, \ldots, x_{n-1})$, where $\mathbf{Lin}_1$ 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 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 $E = \mathbf{Lin}_2(z_0, z_1, \ldots, z_{6s-1}, x_{3s}, x_{3s+1}, \ldots, x_{n-1})$ as a set of linear functions on X and the first $6s$ z_i's. Iterating upto all the 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

$$Y = \mathbf{Lin}(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem}). \tag{6}$$

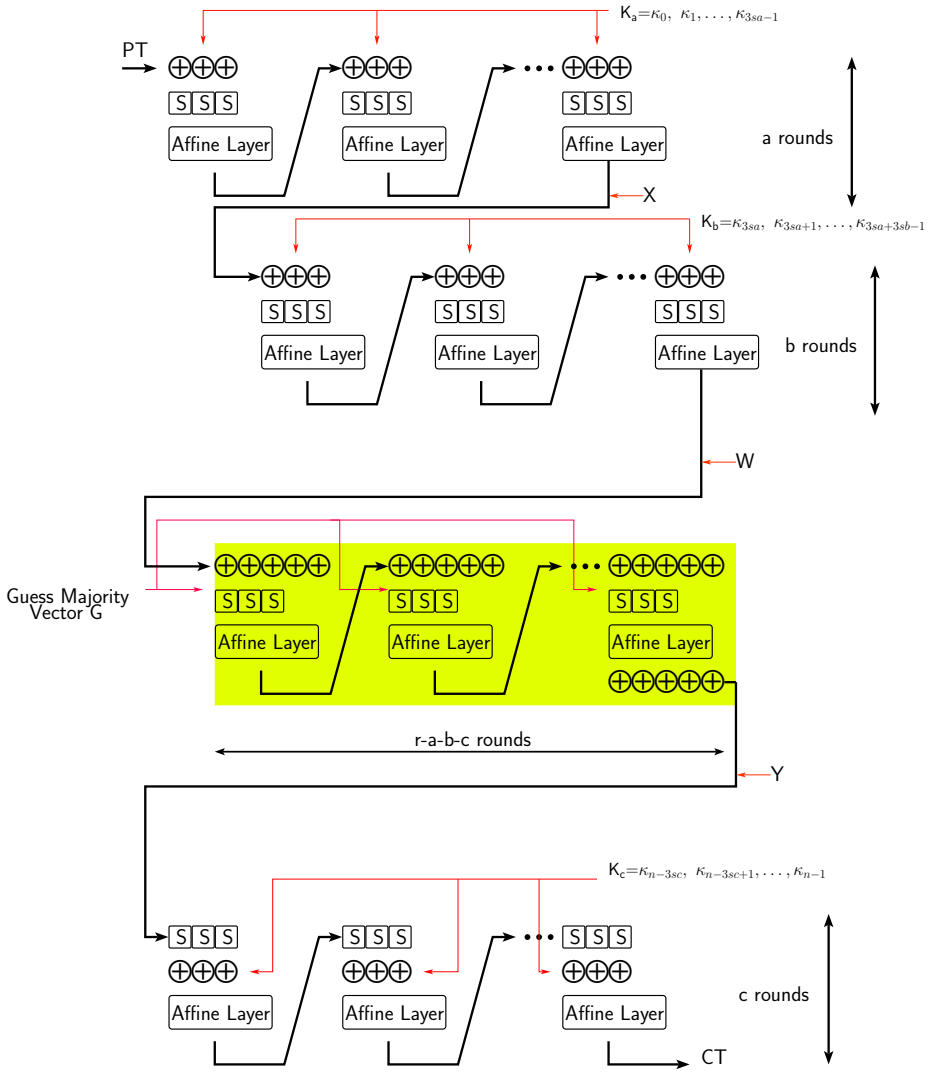


Fig. 5: Splitting LowMC into 4 sections

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 given in Equation (6). Apart from the θ variables the system has n (for Y) $+ (n-3s)$ (for X) $+ (3as+3cs)$ (for K_a, K_c) $= 2n+3(a+c-1)s$ variables. So the above system can be written in matrix notation as $\mathbb{M} \cdot \mathbf{v} = \mathbf{a}$, where $\mathbf{v}$ is the set of $2n+3(a+c-1)s+\theta = (3n+3sb-3s)$ variables, $\mathbb{M}$ is a matrix over GF(2) of size $n \times (3n+3sb-3s)$, and $\mathbf{a}$ is a constant vector. Rearrange $\mathbf{v}$ so that the variables in Λ are the first θ elements of $\mathbf{v}$. Then we use Gaussian elimination to sweep out at least the first θ columns of $\mathbb{M}$. Then 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 $\mathbf{Aff}_1(K_a, X) = \mathbf{Aff}_2(K_c, Y)$, where $\mathbf{Aff}_1, \mathbf{Aff}_2$ are the set of $3sa+3sc-3sb$ affine functions on K_a, X and K_c, Y respectively, obtained above. 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

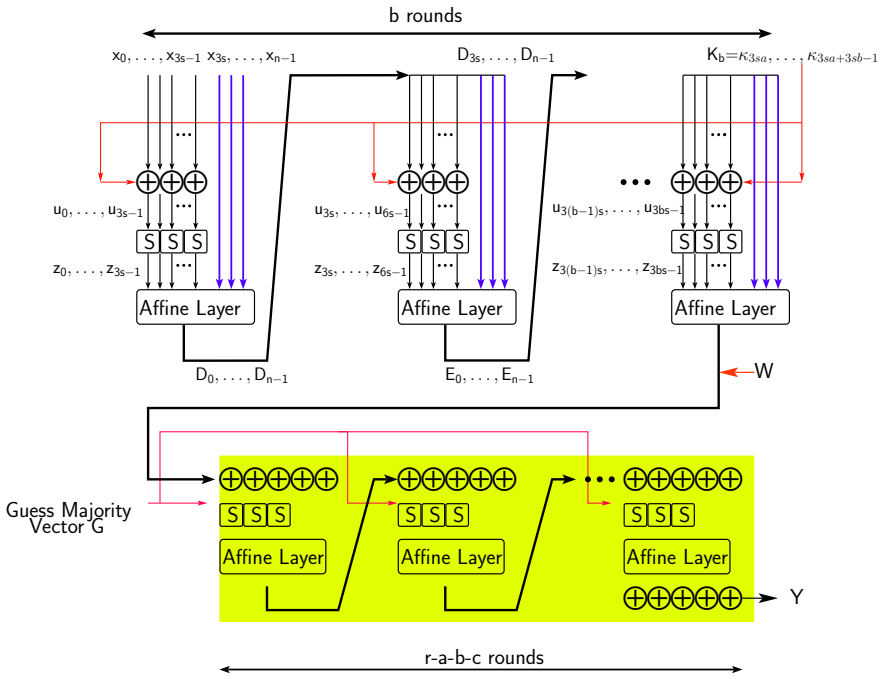


Fig. 6: The middle $b+d$ rounds

guess K_c we can similarly compute Y, by computing backward the last c rounds from the ciphertext. Hence for all the 2^{3sa} values of K_a we make the first list L_1 that contains all the $(3sa - 3sb + 3sc)$-bit vectors calculated from $\mathbf{Aff}_1(K_a, X)$. Similarly for all the 2^{3sc} values of K_c we make the second list L_2 that contains all the $3sa-3sb+3sc$-bit vectors calculated from $\mathbf{Aff}_2(K_c, Y)$. 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 Fig. 6, we can see that $u_i = x_i + \kappa_{3sa+i}$, $\forall i \in [0, 3s-1]$. For the second round we have

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

where $\mathbf{Lin}_{1,i}$ is the i-th linear function of $\mathbf{Lin}_1$ described above. 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

$$\begin{aligned} u_{6s+i} &= E_i + \kappa_{3sa+6s+i},\ \forall\ i \in [0, 3s-1] \\ &= \mathbf{Lin}_{2,i}(z_0, \ldots, z_{6s-1}, x_{3s}, \ldots, x_{n-1}) + \kappa_{3sa+6s+i},\ \forall\ i \in [0, 3s-1] \end{aligned}$$

where $\mathbf{Lin}_{2,i}$ is similarly the i-th linear function of $\mathbf{Lin}_2$. Iterating over all the b rounds we can write the vector equation, $U = K_b + \mathbf{P}(Z, x_{3s}, \ldots, x_{n-1})$, where $\mathbf{P}$ denotes the set of $3bs$ linear expressions obtained by putting together the linear expressions $\mathbf{Lin}_{1,i}, \mathbf{Lin}_{2,i}$ etc. We can now replace K_b in Equation (6) to get

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

This time we 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 $\mathbf{Aff}_3(Z, U) = \mathbf{Aff}_4(X, K_a, K_c)$, where $\mathbf{Aff}_3, \mathbf{Aff}_4$ 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 $\mathbf{Aff}_3(Z, U)$ for each of 2^{3bs} values of Z. In L_4 we store the $3s(a+b+c)$-bit vector given by the expressions $\mathbf{Aff}_4(X, K_a, K_c)$ 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 $2^{3bs+3bs-3s(a+b+c)} = 2^{3sb-3sa-3sc}$. However 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 $K_b = U + \mathbf{P}(Z, x_{3s}, \ldots, x_{n-1})$. 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 $\mathbf{Aff}_{rem}(W, K_{rem}) = Y$ 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 restart 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 formally state 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 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 $Y = \mathbf{Lin}(Z, x_{3s}, \ldots, x_{n-1}, K_a, K_b, K_c, K_{rem})$
   - Eliminate K_b, K_{rem}, Z from the relation and form and equation of form $\mathbf{Aff}_1(K_a, X) = \mathbf{Aff}_2(K_c, Y)$.
   - By exhausting all possible values of K_a keep a list of $\mathbf{Aff}_1(K_a, X)$, where X is computed knowing K_a and plaintext pt.
   - Try all possible values of K_c and find collisions between $\mathbf{Aff}_2(K_c, Y)$ and the list computed in the previous step. Keep a list L of (K_a, K_c) values satisfying the condition.

   Second MITM:
   - Compute the relation $Y = \mathbf{Lin}'(Z, x_{3s}, x_{3s+1}, \ldots, x_{n-1}, K_a, U, K_c, K_{rem})$ by replacing K_b.
   - Eliminate K_b, K_{rem} to get a relation of form $\mathbf{Aff}_3(Z, U) = \mathbf{Aff}_4(X, K_a, K_c)$.
   - For every pair (K_a, K_c) in the list L, compute $\mathbf{Aff}_4(X, K_a, K_c)$.
   - For every possible value of Z, compute $\mathbf{Aff}_3(Z, U)$, where U can be computed efficiently from Z, and look for occurrence with $\mathbf{Aff}_3(Z, U)$ 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 $\mathbf{Aff}_{rem}(K_{rem}, K_a, K_b, K_c, W) = Y$.
   - 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: Before we state our analysis to calculate the computational complexity, let us state a few observations:

1. Note that the number of variables on the right side of Equation (6) is $2n + 3sb - 3s$. Hence using the basis vector logic, forming Equation (6) is equivalent to $2n + 3sb - 3s$ encryptions limited to $r - a - c$ rounds, hence equivalent to $(2n + 3sb - 3s) \cdot \frac{(r-a-c)}{r}$ encryptions.

2. For the first MITM, eliminating $\theta = n - 3s(a - b + c)$ variables in an $n \times (3n + 3sb - 3s)$ matrix using the sweeping out method costs around $\frac{n \cdot \theta \cdot (3n+3sb-3s)}{2rn^2}$ encryptions.

3. Computing U from X and K is equivalent to the encryption of $2n$ base vectors (for the n bits of X and the n bits of K) in b rounds instead of r. So, this costs $2n \cdot \frac{b}{r}$ encryptions

4. For the 2nd MITM, eliminating $3sb$ (K_b) and $n - 3s(a + b + c)$ (K_{rem}) variables in a $n \times (3n + 6sb - 3s)$ matrix requires $\frac{(n-3s(a+c)) \cdot n \cdot (3n+6sb-3s)}{2rn^2}$ encryptions.

5. Solve the system of linear equations to get K_{rem} from $\mathbf{Aff}_{rem}(K_{rem}, K_a, K_b, K_c, W) = Y$. This requires one Gaussian Elimination which is equivalent to $\frac{(n-3s(a+b+c))^3}{2rn^2}$ encryptions.

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. Note that to evaluate $\mathbf{Aff}_1(K_a, X)$ we need to evaluate the first a encryption rounds to get X from the plaintext. Thereafter we evaluate $(3sa - 3sb + 3sc)$ linear expressions in $(3sa + n)$ bits of K_a, X, which requires around $(3sa + 3sb - 3sc) \cdot (3sa + n)$ bit-operations. Similarly to evaluate $\mathbf{Aff}_2(K_c, Y)$ we need to evaluate the last c decryption rounds to get Y from the ciphertext, followed by evaluation of linear expressions that take $(3sa+3sb-3sc) \cdot (3sc+n)$ bit-operations. Hence the first MITM takes time equivalent to $T_1 = \left(\frac{a}{r} + \frac{(3sa-3sb+3sc) \cdot (3sa+n)}{2rn^2}\right) \cdot 2^{3sa} + \left(\frac{c}{r} + \frac{(3sa-3sb+3sc) \cdot (3sc+n)}{2rn^2}\right) \cdot 2^{3sc}$ 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 $\mathbf{Aff}_3(Z, U)$ and $\mathbf{Aff}_4(X, K_a, K_c)$ having values of K_a, K_c and Z. Computing the value of U from Z takes time less than required in the b encryption rounds. Thereafter, evaluating $3s(a+b+c)$ linear expressions in $6bs$ bits requires $3s(a+b+c) \cdot 6bs$ bit-operations. Again for $\mathbf{Aff}_4$ computing X from K_a requires evaluating the first a encryption rounds. Then evaluation of linear expressions requires $3s(a + b + c) \cdot (3sa + 3sc + n)$ bit-operations. Hence the 2nd MITM takes $T_2 = \left(\frac{b}{r} + \frac{(3sa+3sb+3sc) \cdot (6bs)}{2rn^2} + \frac{a}{r} + \frac{(3sa+3sb+3sc) \cdot (3sa+3sc+n)}{2rn^2}\right) \cdot 2^{3sb}$ encryptions. The expected number of collisions in this procedure is $2^{3sb-3sa-3sc}$ which the attacker needs to filter whenever it is greater than 1. Hence the total

complexity of the attack is estimated as:

$$2^{s(r-a-b-c)} \times \Big[\, T_1 + T_2 \text{ (The 2 MITMs) } + (2^{3s(b-a-c)}) \text{ (Filter Solutions)} + (2n + 3sb - 3s) \cdot \frac{(r-a-c)}{r} + \frac{n \cdot \theta \cdot (3n + 3sb - 3s)}{2rn^2} + 2n \cdot \frac{b}{r} + \frac{(n - 3s(a+c)) \cdot n \cdot (3n + 6sb - 3s)}{2rn^2} + \frac{(n - 3s(a+b+c))^3}{2rn^2} \Big].$$

As n and s go to infinity, the optimal parameters become $a = b = c = 1$ and the asymptotic complexity is equivalent to $\frac{4}{r} * 2^{sr}$, which is an improvement by a factor $n/8$ compared to the linearization attack. When s remains small (e.g. $s = 1$), the optimal parameters can be larger. With $a = b = c = \frac{\log_2(2n)}{3s}$, the complexity is asymptotically $\frac{4\log_2(n)}{3sr} \cdot 2^{sr}$. If we take $sr = n$, this is better than exhaustive search by a factor $\Omega\left(\frac{n}{\log(n)}\right)$. The memory complexity is dominated by the space required for the 2 MITM stages. It can be seen that the total memory complexity in bits can be computed as

$$(3sa - 3sb + 3sc) \cdot (2^{3as} + 2^{3cs}) + (3sa + 3sb + 3sc) \cdot 2^{3bs+1}.$$

For the $\lfloor \frac{n}{s} \rfloor$-round instances, we get the following results. For $n = 128,\ s = 1, r = 128$, if we take $a = b = c = 5$, we get the total complexity around 2^{125} encryptions with 2^{22} bits of memory. For $n = 128,\ s = 10, r = 12$, if we take $a = b = c = 1$, we get the total complexity around 2^{119} encryptions with 2^{38} bits of memory. For the $0.8 \times \lfloor \frac{n}{s} \rfloor$-round instances, we get the following results. For $n = 128,\ s = 1, r = 103$, if we take $a = b = c = 5$, we get the total complexity around 2^{101} encryptions. For $n = 128,\ s = 10, r = 10$, if we take $a = b = c = 1$, we get the total complexity around 2^{99} encryptions. The memory complexity is the same as the corresponding $\lfloor \frac{n}{s} \rfloor$-round attacks.

5.1 Speed-up using Gray-Codes

Note that the technique outlined in [BCC+10] to evaluate a function over all points of its input domain, works best for linear or quadratic functions. As such, it is best to employ the attack when the set of functions for which we want to evaluate over the input space is quadratic/linear. This is only possible if we restrict $a = c = 1$. Let us see why. The first MITM procedure finds collision between two lists using the equation $\mathbf{Aff}_1(K_a, X) = \mathbf{Aff}_2(K_c, Y)$. Note that, thus far, X (rep. Y) has been computed from the plaintext (resp. ciphertext) by guessing K_a (resp. K_c) and evaluating the first a rounds in the forward direction (resp. last c rounds in the backward direction). In order to apply Gray-code based speed-up we need to express X and Y as functions of K_a and K_c. These functions happen to be quadratic only when $a = c = 1$. This condition automatically ensures that in the second MITM equations are also quadratic. This is true since

the second MITM essentially equates $\mathbf{Aff}_3(Z,U) = \mathbf{Aff}_4(X,K_a,K_c)$, and we know that the relation between $U,\ Z$ is quadratic since these are the input-output bits of the LowMC S-box in the middle b rounds. However note that unlike, in the MITM for the complete non-linear layers, there is no pre-computation in the first MITM that helps us reduce the steps in the second MITM. $\mathbf{Aff}_4(X,K_a,K_c)$ needs to be only evaluated for the 2^{3sb} pairs of K_a,K_c that survive the 1st MITM. However to employ Gray-code based speed up we need to evaluate $\mathbf{Aff}_4$ over all points of its input space. We could split $\mathbf{Aff}_4$ into $\mathbf{Aff}_5(K_a,X)+\mathbf{Aff}_6(K_c)$ and then evaluate each of the $\mathbf{Aff}_5$ and $\mathbf{Aff}_6$ separately. Thus the time required for the first MITM would be $T_{G_1} = \frac{3sa-3sb+3sc}{2rn^2} \cdot (2^{3as+1} + 2^{3sc+1})$ encryptions. The 2nd MITM requires $T_{G_2} = \frac{3sa+3sb+3sc}{2rn^2} \cdot (2^{3bs+1} + 2^{3as+1} + 2^{3sc})$ encryptions.

It only makes sense to employ Gray-codes if $T_{G_1} + T_{G_2} < T_1 + T_2$. For $s = 1$, the optimal values of a, b, c are considerably higher and it does not make sense to attempt the Gray-code speed-up using this algorithm. In fact even if we attempt to use this method by forcing $a = b = c = 1$, the complexity is many times higher. Intuitively this makes sense, if $a,\ c$ and s are both 1 then the lists require exhaustive search over only $3as = 3sc = 3$ variables, for which employing even a non-Gray-code approach would take only 2^3 function evaluations. However when $s = 10$, using such Gray-codes to execute the MITM stages is beneficial. For $n = 128,\ s = 10, r = \lfloor \frac{n}{s} \rfloor = 12$, if we take $a = b = c = 1$, we get the total complexity around $2^{110.6}$ encryptions which is better than the previous estimate by a factor of around 2^9. For $r = 0.8 \times \lfloor \frac{n}{s} \rfloor = 10$ using the same parameters we get the total complexity around $2^{90.8}$ encryptions which again outperforms the previous estimate by a factor of around 2^8.

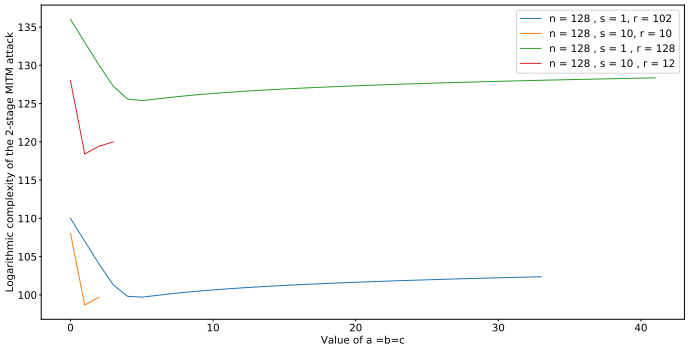


Fig. 7: The base 2 logarithm of the complexity of the 2-stage MITM attack when $n = 128$ and $s = 1, 10$, for $n = 0.8 \times \lfloor \frac{n}{s} \rfloor$, when a, b, c are kept equal and varied.

6 Experimental Results

In this section we present experimental data to showcase how our new attacks stack up in comparison to the attacks proposed in [BBDV20] on instances of LowMC with smaller blocksizes. Our results indicate that for all instances targeted in our paper, there is a significant speedup compared to the previous attacks. Moreover, we provide experimental evidence that our attacks successfully recover the key with a better complexity than exhaustive search for both 3-round with full S-box layer and n/s-round with partial S-box layer variants.

All the attacks and variants of the encryption function were implemented in Sage and ran on an Intel Xeon E5-2680 processor with 256 GB of memory. Each attack was run for several randomly generated instances. The complexity figures are reported by computing the base 2 logarithm of the amount of time taken by the attack, divided by the amount of time one encryption takes.[1]

Full S-box Layer: For the 2-round full S-box layer variant of the cipher, we implemented all three Linearization, 2-step MITM and 2-step MITM with gray-code enumeration attacks for $n = 18$. The results are presented in Fig. 8. On average, the linearization attack required $2^{16.38}$ encryptions to recover the key, where as the 2-stage MITM, and the gray code enumeration attacks required $2^{13.31}$ and $2^{6.42}$ encryptions to yield a solution respectively.

We also implemented the attack using Gray-code enumeration for 3-round variants of block size 12. Fig. 9 show cases the complexity of this attack for several randomly generated samples. Our experimental results indicate that the 3-round variant of this attack yields a solution faster than exhaustive search for all the samples we ran the attack for and the average complexity of our experiments was $2^{5.88}$ encryptions for $n = 12, s = 4, r = 3$.

Partial Non-Linear Layer: For the partial S-box layer variant of the cipher with number of rounds equal to $r = \lfloor \frac{n}{s} \rfloor \times 0.8$, we implemented the 2-stage MITM attack described in section 5, the linearization method described in [BBDV20] and in addition the special case gray-code enumeration attack described at the end of 5. For $n = 16$, $s = 1$ and $r = 12$ the linearization attack yielded a complexity of $2^{10.29}$ encryptions, and the two-step MITM and the gray-code enumeration attacks yielded a solution in $2^{8.46}$ and $2^{8.50}$ encryptions respectively.

For the 2-step MITM attack, we ran the experiments for 3 instances of $a = b = c = 1$, $a = b = c = 2$ and $a = b = c = 3$. According to our experimental results the best performance was when $a = b = c = 1$. The results of the 3 attacks are demonstrated in Fig. 10a, and it is evident that both our new attacks are significantly faster than the linearization method.

We also experimented the attack for $n = 12$, $s = 1$ and $r = n/s = 12$ and $a = b = c = 1$. According to our experimental results demonstrated in Fig. 10b,

[1] The source code of the attacks can be found at https://gitlab.epfl.ch/barooti/lowmc-challenge-round-3

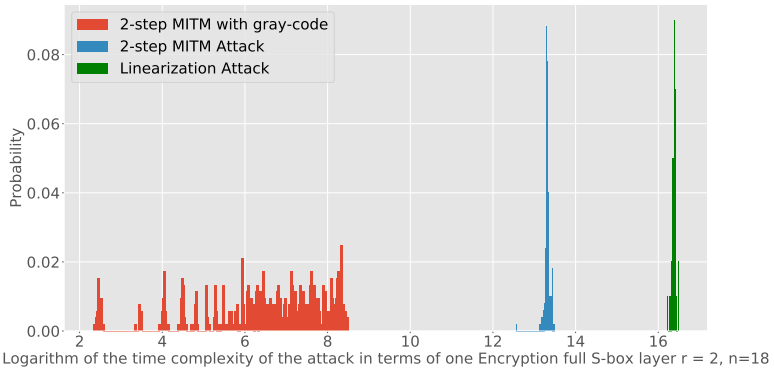


Fig. 8: The histogram of base 2 logarithm of the time complexity of all linearization, 2-stage MITM and 2-stage MITM with gray-code enumeration attacks for $n = 18$, $s = 6$, $r = 2$, in terms of the time it takes to perform a single encryption with the same key, the same affine layers, and the same key update functions.

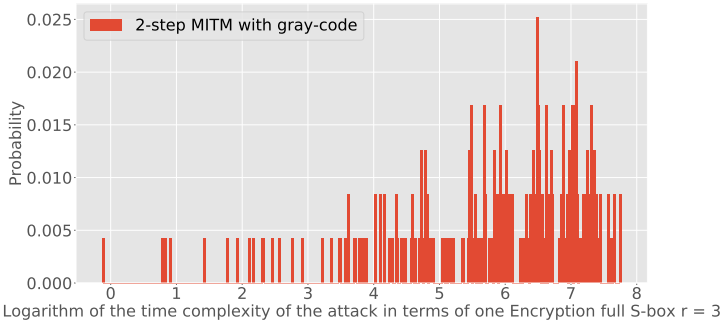
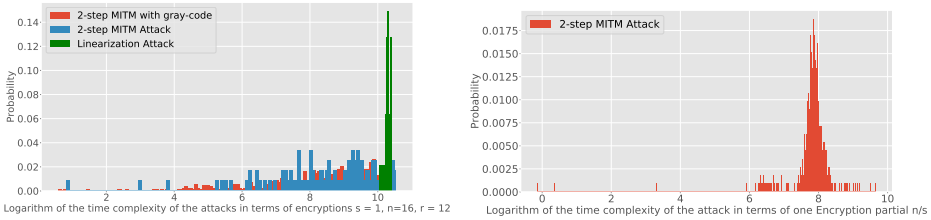


Fig. 9: The histogram of base 2 logarithm of the time complexity of the gray-code enumerated 2-stage MITM attack for $n = 12$, $s = 4$, $r = 3$, in terms of the time it takes to perform a single encryption

this attack had an average complexity of $2^{7.402}$ encryptions, indicating a speed up over exhaustive search.



(a) (b)

Fig. 10: (a) The logarithm of the complexity of 2-step MITM, 2-step MITM with gray-code enumeration and linearization attacks for the partial S-box layer variant with parameters $n = 16$, $s = 1$ and $r = 12$, (b) The logarithm of the complexity of the two-step MITM attack for $n = 12, s = 1, r = 12$.

7 Conclusion

In this paper, we present a 2-stage MITM on several instances of LowMC using only a single plaintext/ciphertext. The first MITM stage reduces the key candidates corresponding to a fraction of key bits of the master key. The second MITM stage between this reduced candidate set and the remaining fraction of key bits successfully recovers the master key. We have shown with experimental evidence on smaller versions of LowMC that the combined computational complexity of both these stages is significantly lower than those reported in [BBDV20].

Acknowledgments: Subhadeep Banik was supported by the Swiss National Science Foundation (SNSF) through the Ambizione Grant `PZ00P2_179921`. Khashayar Barooti was supported by the SNSF through the project grant `192364` on Post Quantum Cryptography.

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.

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.

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.

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.

Din. Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation system over gf(2). IACR Cryptology ePrint Archive, 2021/578. Available at https://eprint.iacr.org/2021/578.pdf.

DKP+19. Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. Linear equivalence of block ciphers with partial non-linear layers: Application to lowmc. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2019.

DLMW15. Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on lowmc. In Tetsu Iwata and Jung Hee Cheon, editors, *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*, volume 9453 of *Lecture Notes in Computer Science*, pages 535–560. Springer, 2015.

DN19. Itai Dinur and Niv Nadler. Multi-target attacks on the picnic signature scheme and related protocols. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 699–727. Springer, 2019.

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.

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.

LIM21. Fukang Liu, Takanori Isobe, and Willi Meier. A simple algebraic attack on 3-round lowmc. *IACR Cryptol. ePrint Arch.*, 2021:255, 2021.

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.

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.