Solving LowMC Challenge

Subhadeep Banik[1], Khashayar Barooti[1], F. Betül Durak[2], Serge Vaudenay[1]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland
{serge.vaudenay,khashayar.barooti,subhadeep.banik}@epfl.ch
[2] Robert Bosch LLC - Research and Technology Center - Pittsburgh PA, USA
durakfbetul@gmail.com

1 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. $f = x_0 \cdot x_1 + x_1 \cdot x_2 + x_0 \cdot x_2$. It is easy then to see that 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, 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

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

It is easy to see that 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. 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

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

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. A small computer exercise shows us that all functions f for which $\mathbf{M}$ is invertible, are exactly the functions that are balanced. This completes the proof. □

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

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

2 The LowMC challenge

The LowMC round function is 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 by 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 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 number of times to give the ciphertext. From the point of view of cryptanalysis,

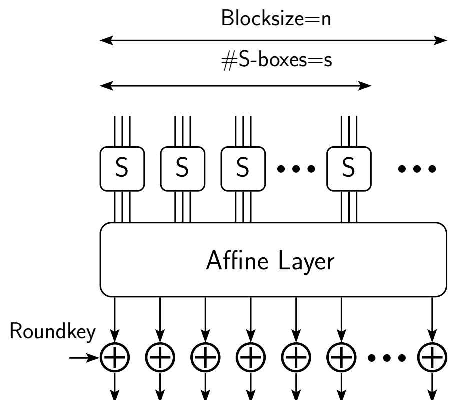

Fig. 1: LowMC Round Function

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 as given in `https://lowmcchallenge.github.io/` specifies 9 challenge scenarios for key recovery given only 1 plaintext-ciphertext pair, i.e. the data complexity $d = 1$.

- $n = 128, \quad s = 1$
- $n = 128, \quad s = 10$
- $n = 129, \quad s = 43$ (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 2,3,4 and for instances with a partial S-box layer can vary between $0.8 * \lfloor \frac{n}{s} \rfloor$, $\lfloor \frac{n}{s} \rfloor$ and $1.2 * \lfloor \frac{n}{s} \rfloor$. The key length k for all instances is n bits.

3 Cryptanalysis by Linearization

A trivial way 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 $= 0.8 * \lfloor \frac{n}{s} \rfloor$.

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

Instance	n	s	r	Type of Attack	Complexity	Section
Full S-box layer	129	43	2	Linearization	2^{86} Enc	3
	192	64			2^{128} Enc	
	255	85			2^{170} Enc	
Partial Sbox layer	128	1	$0.8 * \lfloor \frac{n}{s} \rfloor$	Linearization	$2^{104.4}$ Enc	3
	192	1			$2^{153.6}$ Enc	
	256	1			$2^{204.8}$ Enc	
Partial Sbox layer	128	10	$0.8 * \lfloor \frac{n}{s} \rfloor$	Linearization	2^{100} Enc	3
	192	10			2^{150} Enc	
	256	10			2^{200} Enc	
Full S-box layer	129	43	2	Linearization + MITM	2^{108}	4
	192	64			2^{161}	
	255	85			2^{213}	
Partial Sbox layer	128	1	$0.8 * \lfloor \frac{n}{s} \rfloor$	Linearization+ MITM	$2^{123.7}$	4
	192	1			$2^{185.6}$	
	256	1			$2^{247.5}$	
Partial Sbox layer	128	10	$0.8 * \lfloor \frac{n}{s} \rfloor$	Linearization + MITM	$2^{123.7}$	4
	192	10			$2^{185.6}$	
	256	10			$2^{247.5}$	

Table 1: Summary of results. Note for the Linearization+MITM approach the complexity is given in "evaluations of a quadratic expression".

relating the plaintext and ciphertext becomes a linear expression in the key variables, i.e. of the form

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

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 + 2^t encryption function calculations. For example this is 2^{86} for the LowMC instance with $n = 129, s = 43, r = 2$. Note that a 2 (resp. $0.8 * \lfloor \frac{n}{s} \rfloor$) round encryption of LowMC requires 4 (resp. $1.6 * \lfloor \frac{n}{s} \rfloor$) matrix multiplications over $GF(2)$ (two each in the round function and key update). Since the cost of matrix multiplication and gaussian elimination are of the same order, the computational complexity of a gaussian elimination is certainly less than a 2 (resp. $0.8 * \lfloor \frac{n}{s} \rfloor$)-round LowMC encryption.

3.1 Bypassing the cost of Gaussian elimination

The attack in the previous section requires a gaussian elimination step for each guess of the majority bits. We can bypass this extra computational step by some further modifications, for the 2 round full S-box version. So 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 key which are of course linear functions of the original key $K = RK_0$. We have

$$x = R_1(pt + RK_0, RK_1) = R_2^{-1}(ct, RK_2)$$

Note that the S-box and its inverse are functions can be written in matrix notation as (where f is the majority of x_0, x_1, x_2)

$$S(x) = \begin{bmatrix} 1 & f & f \\ 1+f & 1 & f \\ 1+f & 1+f & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} f \\ f \\ f \end{bmatrix}$$

$$S^{-1}(x) = \begin{bmatrix} 1 & 1+f & f \\ f & 1 & f \\ 1+f & 1+f & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} f \\ f \\ f \end{bmatrix}$$

Consider the forward and inverse round functions R_1 and R_2^{-1}. Let α_i ($i \in [1, s]$) be the majority values at the input of the s-box layer of R_1 and let η_i ($i \in [1, s]$) be the majority values at the input of the inverse s-box layer of R_2^{-1}. Then we can compute x by executing R_1 on the plaintext, also by executing R_2^{-1} on the ciphertext. Thus x can be written as

$$x = M \cdot K + C = N \cdot K + D$$

where the vector $K = [k_0, k_1, \ldots, k_{n-1}]^T$ and M is an $n \times n$ matrix in the ring $GF(2)[\alpha_1, \alpha_2, \ldots, \alpha_s]$, C is an $n \times 1$ vector in the same ring. Also N is an $n \times n$ matrix in the ring $GF(2)[\eta_1, \eta_2, \ldots, \eta_s]$, and D is a $n \times 1$ vector in the above ring. Note that all entries in M, C, N, D are affine polynomials in the respective ring variables. C and D will of course be a function of the plaintext and ciphertext respectively. Rearranging terms we can write

$$(M + N) \cdot K = C + D \Rightarrow \mathbf{A} \cdot K = \mathbf{B}$$

where the matrix $\mathbf{A} = M + N$ and the vector $\mathbf{B} = C + D$ have affine entries in $GF(2)[\alpha_1, \alpha_2, \ldots, \alpha_s, \eta_1, \eta_2, \ldots, \eta_s]$.

We would ideally like to something similar to gaussian elimination to solve this system, but $\mathbf{A}$ has entries in a polynomial ring and not a field, and so we can not directly apply gaussian elimination. However we can try to reduce $\mathbf{A}$

to an echelon form while still in symbolic form. We make use of the procedure $\mathsf{Solve}(\mathbf{A}, \mathbf{B}, i, \mathbb{E})$ described below. We run the instance $\mathsf{Solve}(\mathbf{A}, \mathbf{B}, 0, \emptyset)$

```
Solve($\mathbf{A}, \mathbf{B}, i, \mathbb{E}$)
    Input: $A = [A_{i,j}]$: $n \times n$ matrix over a polynomial ring, $B = [B_i]$: $n \times 1$
           vector over same ring.
    Input: $i$: Integer $\in [0, n-1]$, $\mathbb{E}$: A set containing linear constraints.
    Output: Solution of the system $\mathbf{A} \cdot K = \mathbf{B}$
if $i \geq n$ then
    Find the solution of $\mathbf{A} \cdot K = \mathbf{B}$ as system should be in echelon form.
    Verify that all the constraints in $\mathbb{E}$ are satisfied.
    if All constraints are satisfied then
        Print $K$
    end
    else
        Return $\perp$
    end
end
if $\mathbf{A}_{i,j} \neq 0$ for some $j$ then
    if $\mathbf{A}_{i,j} = 1$ for some $j$ then
        Set $j_0 \leftarrow j$
        $\mathbf{A}', \mathbf{B}' \leftarrow$ Pivot($\mathbf{A}, \mathbf{B}, i, j_0$)
        Solve($\mathbf{A}', \mathbf{B}', i+1, \mathbb{E}$)
        Return
    end
    else
        Set $j_0 \leftarrow j$ such that $\mathbf{A}_{i,j} \neq 0$
        Set $u$ to be a variable in $\mathbf{A}_{i,j_0}$
        Formulate the equation $Eq \Rightarrow f = 0$
        $(\mathbf{A}', \mathbf{B}') \leftarrow$ substitute $f$ in $\mathbf{A}, \mathbf{B}$ as per $Eq$
        Solve($\mathbf{A}', \mathbf{B}', i, \mathbb{E} \cup Eq$)
        Formulate the equation $Eq' \Rightarrow f = 1$
        $(\mathbf{A}', \mathbf{B}') \leftarrow$ substitute $f$ in $\mathbf{A}, \mathbf{B}$ as per $Eq'$
        Solve($\mathbf{A}', \mathbf{B}', i, \mathbb{E} \cup Eq'$)
        Return
    end
end
else
    Solve($\mathbf{A}, \mathbf{B}, i+1, \mathbb{E}$)
end
```

```
Pivot($\mathbf{A}, \mathbf{B}, i, j$)
    Input: $A = [A_{i,j}]$: $n \times n$ matrix over a polynomial ring, $B = [B_i]$: $n \times 1$
           vector over same ring.
    Input: $i$: Integer $\in [0, n-1]$, $j$: Integer $\in [0, n-1]$.
    Output: Modified matrix, vector $\mathbf{A}', \mathbf{B}'$ after applying row addition

Set $c \leftarrow \mathbf{B}'_i \mathbf{B}_i$
for All $v \in [0, n-1]$ do
    $\mathbf{A}'_{i,v} \leftarrow \mathbf{A}_{i,v}$
end
for All $u \in [0, n-1]$ and $u \neq i$ do
    $\mathbf{B}'_u \leftarrow \mathbf{B}_u + \mathbf{A}_{u,j} \cdot c$
    for All $v \in [0, n-1]$ do
        $\mathbf{A}'_{u,v} \leftarrow \mathbf{A}_{u,v} + \mathbf{A}_{u,j} \cdot \mathbf{A}_{i,v}$ /* Add multiple of row*/
    end
end
```

The procedure Solve tries to reduce the matrix equation to an echelon form. Note that if it finds a 1 in any row of the matrix $\mathbf{A}$ (say at position i, j) it immediately calls the procedure Pivot($\mathbf{A}, \mathbf{B}, i, j$), which multiples each row in the matrix by a suitable multiple of the $i - th$ row so that the $j - th$ column is cleared.

Example 1. If the the matrix equation $\mathbf{A} \cdot K = \mathbf{B}$ is given by the following 3×3 matrix and the pivot is the $(1, 1)$ element then Pivot would transform the matrix as follows

$$\begin{bmatrix} a & b & c \\ d & 1 & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \Rightarrow \begin{bmatrix} a + bd & 0 & c + bf \\ d & 1 & f \\ g + hd & 0 & i + hf \end{bmatrix} \cdot \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} c_0 + bc_1 \\ c_1 \\ c_2 + hc_1 \end{bmatrix}$$

Thus the consistency of the equation system is maintained even as a column is cleared.

If we are lucky the Pivot routines clear all the columns of the matrix and there is only a singe 1's left in every row, from which the key can be recovered by evaluating the final vector $\mathbf{B}$ at all possible values of $\alpha_1, \alpha_2 \ldots, \alpha_s$ and $\eta_1, \eta_2 \ldots, \eta_s$. However this is not always the case and sometimes the algorithm will not be able to find any 1 in a given row of the matrix. In that case it has to try all combinations of variables in the matrix entry it chooses as pivot. If the total number of such elements is p, we need a one time cost of 2^p symbolic executions of the Pivot routine.

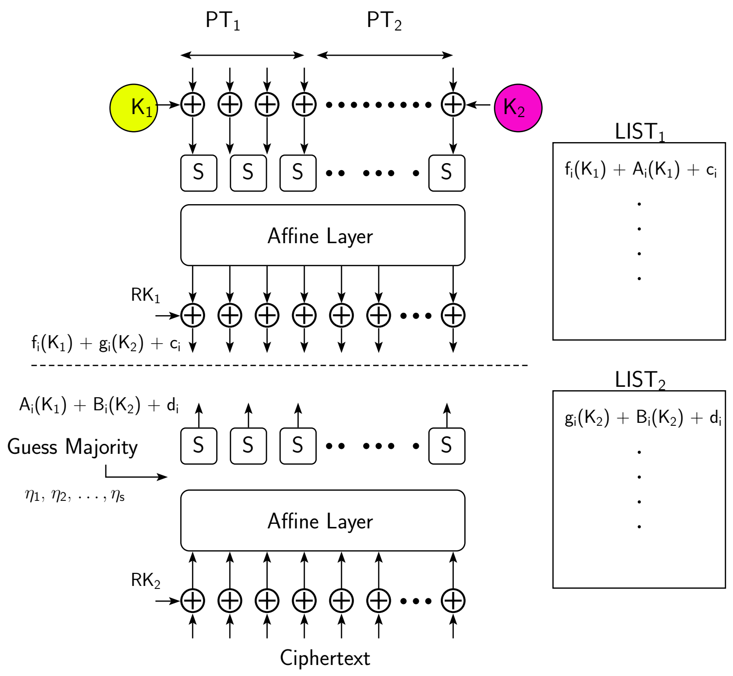
PT$_1$
PT$_2$
K$_1$
K$_2$
S
Affine Layer
RK$_1$
f$_i$(K$_1$) + g$_i$(K$_2$) + c$_i$
LIST$_1$
f$_i$(K$_1$) + A$_i$(K$_1$) + c$_i$
A$_i$(K$_1$) + B$_i$(K$_2$) + d$_i$
Guess Majority
$\eta_1, \eta_2, \ldots, \eta_s$
LIST$_2$
g$_i$(K$_2$) + B$_i$(K$_2$) + d$_i$
RK$_2$
Ciphertext

Fig. 2: Meet in the Middle

4 Meet in the Middle approach

4.1 2 round full S-box layer

In this section 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 $K_1 = [k_1, \ldots, k_t]^T$ and $K_2 = [k_{t+1}, \ldots, k_n]^T$, 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 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. It is easy to see that the same argument holds for the inverse S-box. 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 key 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 $ct \oplus RK_2$. Thereafter we guess the s majority bits $\eta_1, \dots, \eta_s$ at the input of the second round inverse S-boxes to linearize the output. 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, it is easy to see that the bits of K_1 and K_2 are never multiplied in the first round function. 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 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. Given an encryption oracle, we query the all zero plaintext $pt = [0, \dots, 0]$ (or any other plaintext), and receive $ct = [c_0, c_1, \dots, c_{n-1}]$ i.e. the corresponding ciphertext. Take $t = 3 \times \lfloor s/2 \rfloor \approx \frac{n}{2}$. We proceed as follows: we guess the η_i values for all second round S-boxes.

Thereafter in order to perform a key recovery attack one can do 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 $\eta_1, \dots, \eta_s$. 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, and 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 $\eta_1, \dots, \eta_s$.

For each majority guess, the complexity of the attack is dominated by finding a collision between two lists of length $O(2^t)$. So the total complexity of the attack is $O(2^s \times 2^{t+1})$, which for instance for the $n = 129$ bit version is around $2^{43+65} = 2^{108}$.

4.2 MITM on partial S-box layers

In order to perform a MITM on the partial S-box layer instances of LowMC, we rewrite the first r_1 and final r_3 rounds so that the total number of different key

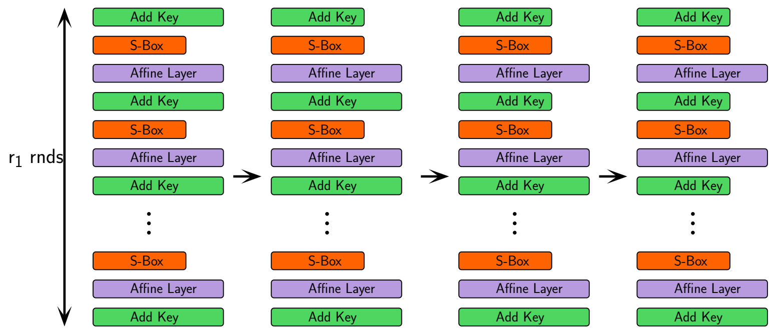

Fig. 3: Transforming the round function in the first r_1 rounds

bits involved in these rounds is $3s$ per round. The transformations are similar to the ones used in [1]. In fact the transform used in the backward direction (see Fig 4) is exactly same as the one used in [1, Fig.1]. The idea is that the affine layer and key addition are interchangeable. Since if L is an 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.

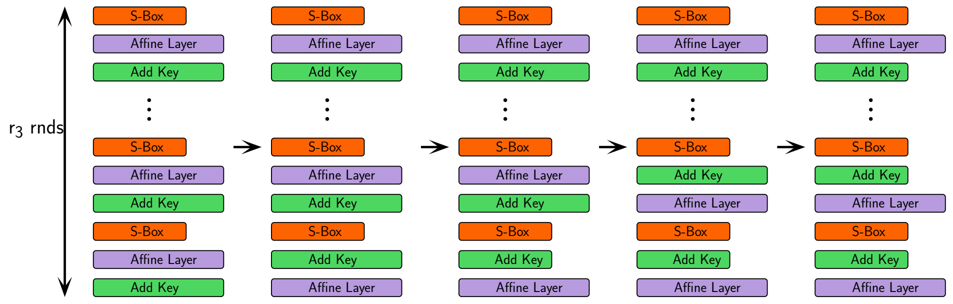

Fig. 4: Transforming the round function in the final r_3 rounds

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 $\kappa_0, \kappa_1, \ldots, \kappa_{n-1}$, it is not difficult to 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 sr_2 majority bits if 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 sr_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 $\{0,1\}^n \to \{0,1\}^n$ and Q_G, W_G are linear functions over $\{0,1\}^{n/2} \to \{0,1\}^n$ 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 sr_2 majority values in the middle rounds. Find the functions L_G, W_G, K_G and C_G. This step is done 2^{sr_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 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 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 $2^{sr_2} \cdot 2^{1+n/2} \approx 2^{sr-n/3+n/2} = 2^{rs+n/6}$. For $r = 0.8\lfloor \frac{n}{s} \rfloor$, this complexity is around $2^{29n/30}$.

References

1. Christian Rechberger, Hadi Soleimany and Tyge Tiessen. Cryptanalysis of Low-Data Instances of Full LowMCv2. In IACR Transactions of Symmetric Cryptology. ISSN 2519-173X, Vol. 2018, No. 3, pp. 163–181.