The Parallel-Cut Meet-In-The-Middle Attack*

Ivica Nikolić, Lei Wang, and Shuang Wu

Nanyang Technological University, Singapore
{inikolic,Wang.Lei,WuShuang}@ntu.edu.sg

Abstract. We propose a new type of meet-in-the-middle attack that splits the cryptographic primitive in parallel to the execution of the operations. The result of the division are two primitives that have smaller input sizes and thus require lower attack complexities. However, the division is not completely independent and the sub-primitives depend (output of one is the input for the other) mutually on a certain number of bits. When the number of such bits is relatively small, we show a technique based on three classical meet-in-the-middle attacks that can recover the secret key of the cipher faster than an exhaustive search. We apply our findings to the lightweight block cipher `Klein` and show attacks on 10/11/13 rounds of `Klein`-64/-80/-96. Our approach requires only one or two pairs of *known* plaintexts and always recovers the secret key.

Key words: meet-in-the-middle attack, cryptanalysis, parallel-cut, block cipher, hash function, Klein

1 Introduction

The meet-in-the-middle attack (MITM) [4] is one of the most popular methods for analysis of ciphers and hash functions. The MITM succeeds when the analyzed primitive can be divided into two parts, each with some independent (of the other part) input bits. More advanced forms of MITM try to exploit the possible division of the primitive in way such that the number of steps covered by the attack is as large as possible. In addition, the division is not always sequential – that is the first part covers some rounds at the beginning and at the end of the primitive, while the second covers some middle rounds. Indeed, this idea forms the basis of the splice-and-cut technique [1] developed by Aoki and Sasaki for finding preimages in hash functions.

What is common for all of the MITM methods, is that they split the initial cipher (or hash function) into two subciphers which are in fact round-reduced versions of the initial cipher – each of the subciphers has the same state size as the cipher, and possibly a smaller key. In this work we propose different division of the cipher into subciphers. Our idea is to split the cipher into two parts that have both smaller key and state sizes. This is achieved by dividing each word (or byte) of the state and the key into two parts and investigating the effects of the

* The work in this paper is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

state (and key schedule) transformations on such division. When the division allows one part of the state to be updated based on the values of the bits of that part and only a few other bits from the other part, then we basically end up with two smaller subciphers which have twice as smaller key and state sizes, and thus require much lower complexity even for a brute attack. We call this approach *parallel-cut meet-in-the-middle attack* to emphasize the fact that the division is parallel to execution flow (see Fig. 1), as each transformation in the cipher is divided into two smaller transformations. In contrast, the classical MITM can be seen as perpendicular-cut, since the division is perpendicular to the execution flow of the whole cipher.

Even with such unorthodox division, no modern cipher allows to be split into subciphers that are completely independent, and there are always bits that are common for both of the subciphers. The division of the parallel-cut should be such that the number of these bits is minimal. We show that the bits indeed can be used to our advantage by presenting approach based on three classical MITM attacks: the first and the second MITMs are used to recover the master keys of the subciphers, and the final, third MITM is used to match the common bits. To make sure that the initial master key can be recovered, the intermediate results of the first two MITMs have to be stored in the table (and later used in the third MITM). Thus at the current stage, our approach requires non-negligible memory. On the other, as in the case of the classical MITM, the data requirement is minimal – one or a few pairs of *known* plaintext-ciphertext.

We apply the idea of parallel-cut meet-in-the-middle attack on the example of the lightweight block cipher `Klein`. The published analysis of this cipher is as follows. The designers have shown a 5-round integral attack with time complexity of 2^{48} [5]. Aumasson *et al.* [2] and Yu *et al.* [10] have independently proposed key recovery attacks for 8-round `Klein`-64 by exploiting the same high probability truncated differential. Aumasson *et al.* used the technique of neutral bits [3] to reduce the complexity and make the attack practical. Yu *et al.* also proposed an integral attack on 7-round `Klein`-64 and 8-round `Klein`-80. At the Rump Session of FSE 2013, Lallemand and Naya-Plasencia [6] announced improved attacks on `Klein`, including full-round attack on `Klein`-64 as well as improved attacks on `Klein`-80 and `Klein`-96.

We show that both the key schedule and the round function of `Klein` can be divided into two independent parts with only a few bit guesses per round. Each such function acts only on the upper (for the first subcipher) or the lower (for the second subcipher) parts of the bytes in `Klein`. This leads to a complete separation of the cipher into two independent subciphers with twice as small key and state sizes and allows to apply our new method. By using a meet-in-the-middle (MITM) approach we recover the key of each subcipher. At the end, we apply another MITM to match the guessed bits and thus to find the master key. We also show that additional improvements of our technique in terms of memory complexity are possible – in general these improvements are not universal, but dependent of the structure of the cipher. Another, cipher specific property given on the example of cryptanalysis of `Klein` is the time-memory tradeoff that can be achieved using our approach. We note that our attacks require only one (or two, in the case of larger keys) pair of *known* plaintext-ciphertext and result in key recoveries for 10/11/13 rounds of `Klein`-64/-80/-96 faster than exhaustive search. To confirm the correctness of our findings, we have implemented the attack on 4 rounds. A comparison of the attacks on `Klein` is shown in Table 1.

Table 1. Comparison of attacks on `Klein`

Key	Rounds	Type	Time	Data	Memory	Source
64	7	Integral	$2^{45.5}$	$2^{34.3}$ CP	2^{24}†	[10]
64	8	Trunc. diff.	$2^{46.8}$	2^{32} CP	2^{16}†	[10]
64	8	Trunc. diff.	2^{35}	2^{34} CP	2^{13}†	[2]
64	12	Trunc. diff.	$2^{60.8}$	$2^{57.2}$ CP	negl	[6]
80	8	Integral	$2^{77.5}$	$2^{34.4}$ CP	2^{24}†	[10]
80	14	Trunc. diff.	2^{77}	2^{44} CP	negl	[6]
96	15	Trunc. diff.	2^{91}	2^{50} CP	negl	[6]
64	10	PC MITM	2^{62}	1 KP	2^{60}	this paper
80	11	PC MITM	2^{74}	2 KP	2^{74}	this paper
96	13	PC MITM	2^{94}	2 KP	2^{82}	this paper
n	r	MITM	$2^{\frac{n}{2}+6\lfloor\frac{r+t}{2}\rfloor}$ $+2^{\frac{n}{2}+6r-32}$ $+2^{n-64}$	2 KP	$2^{6\lfloor\frac{r-t}{2}\rfloor}$ $+2^{6(r-t)+\frac{n}{2}-32}$	this paper

CP: Chosen plaintext
KP: Known plaintext
†: Memory requirement based on our estimation

The paper is structured as follows. In Section 2 we describe our new technique. In Section 3, we give an application of the parallel-cut MITM for the analysis of `Klein`. More advanced techniques, such as partial matching technique (which allows to extend the attack for an additional round), and time-memory tradeoff are presented as well. Finally, Section 4 concludes the paper and proposes a few open problems.

2 The Parallel-Cut MITM

The MITM attack is mostly used to recover the secret key of a cipher and to find preimage of a hash/compression function. In both of the cases, we deal with a cryptographic primitive $F(X,Y)$ that has two inputs X,Y - the input X is known to the attacker and the input Y is unknown. The MITM targets to recover the second unknown input Y, given only a single pair (P,C) of one input and one output, i.e. $F(P,Y)=C$.

The standard MITM attack regards the r-round cryptographic primitive $F(X,Y)$ as a composition of two primitives: the first being r_1-round primitive $F_1(X,Y_1)$, and the second r_1-round $F_2(X,Y_2)$, where $F(X,Y)=F_2(F_1(X,Y_1),Y_2)$,

$r_1 + r_2 = r$, and the inputs Y_1, Y_2 are functions of the input Y. The MITM would succeed if for some $F_1(X, Y_1), F_2(X, Y_2)$, the inputs Y_1, Y_2 have independent bits, i.e. some bits of Y are input only to Y_1 and vice-versa. When $F_2(X, Y_2)$ is invertible and the state size of the primitive is not too large, the MITM can be described as:

1. Fix arbitrary value V of the bits of Y that are input to both F_1, F_2
2. Create a set A of all intermediate state values for F_1 by going through all possible independent bits of Y_1 from the input P, i.e. $A = \{S | S = F_1(V||i, P)\}$
3. Create a set B of all intermediate state values for F_2 by going through all possible independent bits of Y_2 from the output C, i.e. $B = \{S | S = F_2^{-1}(V||j, C)\}$
4. Check on collisions between the sets A, B. A collision corresponds to the target value Y

When the state size is n bits, and there are k_1 independent bits in Y_1 and k_2 in Y_2, the colliding set would have a size of $2^{k_1+k_2-n}$, and can be produced with 2^{k_1} calls to F_1, and 2^{k_2} calls to F_2. In case $k_1 + k_2 < n$ the procedure is repeated $2^{n-k_1-k_2}$ times.

A crucial observation in the above attack is that the primitive F is divided into two sequential parts, that is the division is *perpendicular* to the execution flow. Our variant of MITM cuts the primitive in *parallel* to the execution flow (see Fig. 1) and thus the name parallel-cut MITM (PC MITM). Let us take a closer look at this idea. We assume we have an n-bit block cipher $E_K(P)$ with a pair (P, C) of known plaintext-ciphertext, and we want to find the secret key K. Let $G(S, K_r)$ be the round function of the cipher, where S is the input state and K_r is the round key. If r is the number of rounds in the cipher, then the cipher can be described as $S_{i+1} = G(S_i, K_i)$ and $S_0 = P, S_r = C$. Assume we can split the round function $G(S_r, K_r)$ into two smaller round functions $G_1(S_r^1, K_r^1), G_2(S_r^2, K_r^2)$, such that $G(S_r, K_r) = G_1(S_r^1, K_r^1)||G_2(S_r^2, K_r^2)$, where S_r^1, K_r^1 are the $\frac{n}{2} + a_1$, $\frac{n}{2} + b_1$ leftmost bits, while S_r^2, K_r^2 are the $\frac{n}{2} + a_2$, $\frac{n}{2} + b_2$ rightmost bits of S_r, K_r, respectively (see Fig. 2). That is

$$G_1 : \{0,1\}^{\frac{n}{2}+a_1} \times \{0,1\}^{\frac{n}{2}+b_1} \to \{0,1\}^{\frac{n}{2}} \quad (1)$$

$$G_2 : \{0,1\}^{\frac{n}{2}+a_2} \times \{0,1\}^{\frac{n}{2}+b_2} \to \{0,1\}^{\frac{n}{2}} \quad (2)$$

In other words, each of two halves of the output state, can be determined by the knowledge of one half of the input state and one half of the round key, and a few bits from the other half. If in each round of the cipher, the positions of these bits are the same, then for the whole cipher it follows that the left half of the ciphertext, can be determined from the left half of the plaintext and $r \cdot (a_1 + b_1)$ bits that come from the opposite right half. Similar observation holds for the right half of the ciphertext, but this time the number of bits is $r \cdot (a_2 + b_2)$.

Hence we can split the whole cipher into two subciphers, with twice as small state and key sizes that depend additionally on some known bits. Before we move further we would like to point out that the division of the state and the round keys on left and right half is only to simplify the presentation. The division can

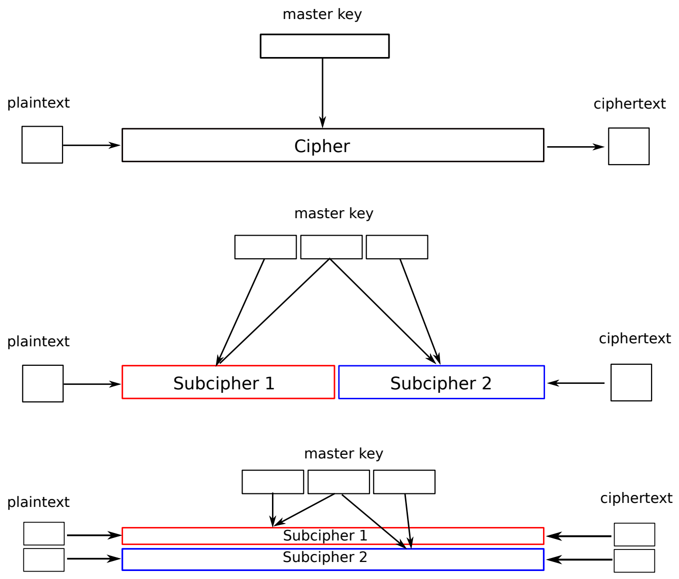

Fig. 1. The flow division in MITM attacks. The original primitive is given at the top, followed by the division used in the standard MITM attacks. At the bottom is the parallel-cut division.

be on any two sets, e.g. the first 10 and the last 20 bits of the state and round keys belong to the first, while the rest belongs to the second cipher. This division is correct[1], as long as both the input and the output states depend on same bits (plus a few guessed).

If the number of guessed bits per round is zero, i.e. $a_i = b_i = 0, i = 1, 2$, then we have divided the cipher into two completely independent ciphers, thus we can easily recover the key (which is now only half of the size of the initial key) independently for each subcipher – this key recovery attack requires only $2 \cdot 2^{\frac{k}{2}}$ effort, for a cipher with k-bit key. However this is not the case for modern ciphers and there are always some unknown bits, i.e. either $a_i > 0$, or $b_i > 0$, or both. Further we assume that $a_i > 0, b_i = 0, i = 1, 2$ as later in the paper we deal with such case only.

Now let us see how to use the guessing bits to our advantage. Let us focus on the first subcipher E_1 that takes as input the left half of the plaintext, the left half of the key, and b_1 guessing bits per round, and outputs the left half of the ciphertext. Thus it is r-round cipher with $\frac{n}{2}$-bit state, $\frac{k}{2}$-bit and additionally depends on $r \cdot b_1$ unknown bits. To find the key used in this subcipher from the known pair of plaintext-ciphertext (note that if $(P, C) = (P_1 || P_2, C_1 || C_2)$ is the known pair for the whole cipher, then (P_1, C_1) is the known pair for this

[1] Actually we are looking for such a division that minimizes the number of guessed bits from the other half.

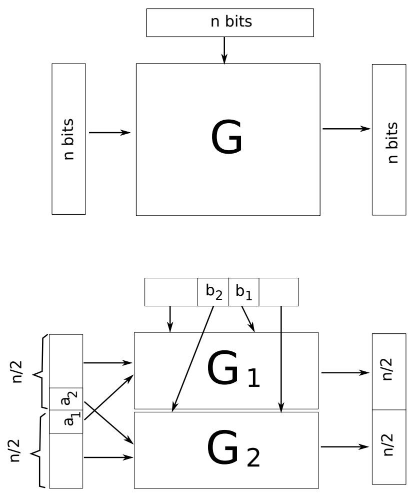

Fig. 2. Division of the round function G (top), into two smaller functions G_1, G_2 (bottom).

subcipher), we actually apply a classical MITM attack[2], where the independent inputs for the MITM are the guessing bits:

1. Fix a key K for the subcipher.
2. Create a set A of states obtained after $\frac{r}{2}$-rounds of encryption of P_1 with all possible values of the guessing bits in of these $\frac{r}{2}$ rounds. The size of A is $2^{\frac{r}{2}\cdot b_1}$.
3. Create a set B of states obtained after $\frac{r}{2}$-rounds of decryption of C_1 with all possible values of the guessing bits in of these $\frac{r}{2}$ rounds. The size of B is $2^{\frac{r}{2}\cdot b_1}$.
4. For each collision between A and B, store in a master table T_1 the key K, the $r \cdot b_1$ values for the guessing bits, as well as the $r \cdot b_2$ values of the bits that the second subcipher depends on.
5. Go to step 1.

Obviously whenever we obtain a collision, we find a a key that encrypts P_1 to C_1 with some specific guessing bits (that come from the other subcipher). The logic behind storing additionally the values of the guessing bits for the second subcipher will be explained later. The size of the state is $\frac{n}{2}$ bits, thus for each key $|A| = |B| = 2^{\frac{r}{2}\cdot b_1}$, and the size of the colliding set is $2^{r\cdot b_1 - \frac{n}{2}}$. Therefore the master table T_1 for all keys will have $2^{\frac{k}{2}+r\cdot b_1-\frac{n}{2}}$ entries. To produce the

[2] Note that a simple exhaustive key search would required $2^{\frac{k}{2}+r\cdot b_2}$ encryption queries, hence when $r \cdot b_2 > \frac{k}{2}$ the exhaustive search of the subcipher has a higher complexity then the exhaustive search of the cipher – therefore we present more advanced technique of finding the key for the subcipher.

set A, B for all keys of the subcipher, and thus the master table T_1, we need $2^{\frac{k}{2}} \cdot 2^{\frac{r}{2} \cdot b_1} = 2^{\frac{k}{2} + \frac{r}{2} \cdot b_1}$ encryption queries.

Next we focus on the second subcipher E_2 and perform exactly the same MITM attack, but this time we use the guessing bits from the first subcipher, i.e. the bits coming from the first subcipher are the guessing bits for the second. We create another master table T_2 for E_2 – the time complexity is $2^{\frac{k}{2} + \frac{r}{2} \cdot b_1}$, and the table has $2^{\frac{k}{2} + r \cdot b_2 - \frac{n}{2}}$ entries. Note that in both T_1 and T_2 each entry consists of a $\frac{k}{2}$-bit key, $r \cdot b_1$ $(r \cdot b_2)$ guesses for the incoming (outcoming) bits, and $r \cdot b_2$ $(r \cdot b_1)$ guesses for outcoming (incoming) bits. The final step of the key recovery attack for the whole cipher E is to find collisions between the master tables T_1, T_2 on the guesses. As there are $r \cdot (b_1 + b_2)$ guesses, we will end up with

$$2^{\frac{k}{2} + r \cdot b_1 - \frac{n}{2}} \cdot 2^{\frac{k}{2} + \frac{r}{2} \cdot b_1} \cdot 2^{-r(b_1 + b_2)} = 2^{k-n},$$

possible keys[3]. If $k > n$ we will need only a few additional plaintext-ciphertext pairs to find the exact key.

In general the above PC MITM can be described as:

1. Split the cipher E into two subciphers E_1, E_2 such that each subcipher depends on as less as possible bits of the other subcipher
2. For each of the subciphers, using a classical MITM recover the key under all possible values of the guessing bits coming from the other subcipher
3. For each pair of recovered keys of the first and the second subcipher, check if the guessing bits correspond – use another MITM to perform this procedure

The success of the PC MITM, in particular the number of rounds that can be attacked, depends on how well the (round-reduced) cipher allows division into subciphers with minimal bit guesses per round. This on the other hand is related to the round diffusion, however a strong diffusion does not necessarily mean a strong resistance against PC MITM. For example, a hypothetical 64-bit cipher that has only modular additions, XORs, and rotations on 32 bits can achieve a full diffusion in one-two rounds, however it can easily be attacked with PC MITM as for each addition only 1 bit coming from the lower half has to be guessed. In the sequel we present a PC MITM attack on `Klein`.

3 Application to `Klein`

3.1 Notations

First, we introduce the notations used in this section. For a byte b, the values b^H and b^L stand for the higher and lower nibbles of b, i.e. $b = b^H || b^L, |b^H| = |b^L| = 4$. The i-th bit of b is denoted as b_i, where b_0 is the least significant bit. Thus $b^H = b_7||b_6||b_5||b_4$, $b^L = b_3||b_2||b_1||b_0$ and $b = b_7||b_6||b_5||b_4||b_3||b_2||b_1||b_0$.

[3] This number is in fact the average number of possible k-bit keys that encrypt n-bit plaintext P to ciphertext C, i.e. with our approach we find all the possible keys for the cipher.

The most significant and least significant bits of b are denoted as $MSB(b)$ and $LSB(b)$ respectively. For example $MSB(b^L) = b_3$ and $LSB(b^H) = b_4$.

Let X be an s-byte vector, i.e. $X = (X[0], \ldots, X[s-1])$. With X^H, X^L we denote the vectors of higher and lower nibbles of X, i.e. $X^H = (X[0]^H, \ldots, X[s-1]^H), X^L = (X[0]^L, \ldots, X[s-1]^L)$. We use the symbol of $||$ to denote concatenation of nibbles within bytes, i.e. $X^H||X^L = X$. In addition, we use $MSB(X)$ to denote the s-bit concatenation of the MSBs in all bytes of the vector X, i.e. $MSB(X) = MSB(X[0])|| \ldots ||MSB(X[s-1]) = X[0]_7|| \ldots ||X[s-1]_7$. For a vector of nibbles such as X^L, $MSB(X^L) = X[0]_3|| \ldots ||X[s-1]_3$.

3.2 Description of `Klein`

`Klein` is a family of 64-bit block ciphers and has three versions: `Klein`-64, `Klein`-80 and `Klein`-96, which use secret keys of 64, 80 and 96 bits and iterate the same round function for 12, 16 and 20 rounds, respectively. The three versions differ only in the key length and essentially in the key schedule. The 64-bit state of `Klein` can be seen both as containing 16 nibbles[4] or 8 bytes (a pair of two consecutive nibbles consists a byte). This dual view is important for the description of the state transformations as well as for our attacks. Obviously, a nibble transformation can always be seen as a byte transformation, but not necessarily the other way around. Thus, we will describe the state transformation as nibble oriented in the cases when this is possible. The i-th round function of `Klein` has four distinct operations:

AddRoundKey : XOR of the 16-**nibble** subkey K_{i-1} to the state.
SubNibbles : Application of an 4-bit Sbox to each **nibble** of the state.
RotateNibbles : Cyclic rotation of the state leftwards by 4 **nibbles**.
MixNibbles : Application of the AES MixColumn to two groups of 4 **bytes**.

Note that only the MixNibbles step is byte-oriented, whereas the other three operations are nibble-oriented. After the last round, another subkey is XORed to the state.

The MixNibbles operation is the diffusion layer and consists of two instances of the AES operation MixColumns which can be expressed as a matrix multiplication in the finite field $GF_{2^8} = F_2[x]/x^8 + x^4 + x^3 + x + 1$:

$$MixColumn(v) = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \cdot \begin{pmatrix} v[0] \\ v[1] \\ v[2] \\ v[3] \end{pmatrix} \tag{3}$$

The key schedule of `Klein` is as follows. At first, the master key denoted as K_0, is split into t bytes $\{K_0[0], K_0[1], ..., K_0[t-1]\}$, where $t = 8, 10$ and 12 for 64-bit, 80-bit, and 96-bit key, respectively. Then the subkey K_i is computed from K_{i-1} for $i = 1, 2, ..., r$.

[4] One nibble is 4 bits.

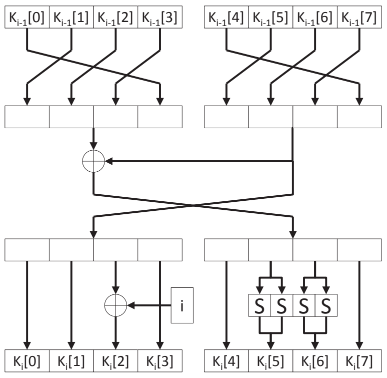

Fig. 3. Key schedule of `Klein`-64

1. Cyclic rotation of the bytes leftward by one position, independently in the left and right halves of K_{i-1}, i.e. rotate both $K^0_{i-1} = (K_{i-1}[0], K_{i-1}[1], ..., K_{i-1}[\frac{t}{2} - 1])$ and $K^1_{i-1} = (K_{i-1}[\frac{t}{2}], K_{i-1}[\frac{t}{2} + 1], ..., K_{i-1}[t - 1])$.
2. Application of a Feistel-like transformation on the left and right halves of K_{i-1}, i.e. $K^0_i = K^1_{i-1}$ and $K^1_i = K^0_{i-1} \oplus K^1_{i-1}$.
3. XOR of the round counter i (one byte) to the third byte of K_i, i.e. $K_i[2] = K_i[2] \oplus i$.
4. Application of four S-boxes to the four nibbles in the second and the third bytes of K^1_i, $(K^1_i[1] = K_i[\frac{t}{2} + 1])$ i.e. $K^1_i[1] = S[K^1_i[1]^H]||S[K^1_i[1]^L]$, $K^1_i[2] = S[K^1_i[2]^H]||S[K^1_i[2]^L]$, where $S[\cdot]$ stands for the 4-bit Sbox.

where $S[\cdot]$ stands for the 4-bit Sbox. The key schedule for `Klein`-64 is shown in Fig. 3.

After each round of the key schedule, only the leftmost 64 bits of K_{i-1} are used as a subkey in the i-th round of the encryption.

3.3 The Parallel-Cut Division of `Klein`

From the specification of `Klein`, we can see that the key schedule, AddRoundKey, SubNibbles and RotateNibbles are all nibble-oriented and do not have diffusion between the higher and lower nibbles within the bytes of the state. This property for the key schedule only, has already been pointed out in [2]. We call the byte transformation $T(X)$ a *nibble-separable*, if it can be expressed as $T(X) = T^H(X^H)||T^L(X^L)$, where $T^H(Y), T^L(Y)$ are nibble transformations. The nibble-separation allows to launch PC MITM. If there were no MixNibbles operations, the higher and lower nibbles would never mix and the cipher could be divided completely into two independent subciphers. Thus let us focus on the MixColumn operation used in MixNibbles of `Klein` and try to find the the diffusion rate between the higher and lower nibbles.

Observation 1 *The operation MixColumn is nibble-separable with an additional dependency on 4 MSB inputs from the opposite halves, i.e. for an input v,* $MixColumns(v) = MixColumns(v)^H || MixColumns(v)^L$*, where*

$$MixColumn(v)^H = MixColumn^H(v^H, MSB(v^L)),$$
$$MixColumn(v)^L = MixColumn^L(v^L, MSB(v^H)).$$

Moreover, the entropy introduced by the MSBs is only 3 bits.

The above observation claims that MixColumns is almost nibble-separable, and the higher/lower output nibbles can be computed from the higher/lower input nibbles and only four additional most significant bits of the lower/higher nibbles. Also, instead of 4-bit entropy, these 4 MSBs introduce only 3-bit entropy.

Proof. First let us rewrite (3) as:

$$MixColumn(v) = \begin{pmatrix} 00\ 01\ 01\ 01 \\ 01\ 00\ 01\ 01 \\ 01\ 01\ 00\ 01 \\ 01\ 01\ 01\ 00 \end{pmatrix} \cdot \begin{pmatrix} v[0] \\ v[1] \\ v[2] \\ v[3] \end{pmatrix} + \begin{pmatrix} 02 \cdot (v[0] + v[1]) \\ 02 \cdot (v[1] + v[2]) \\ 02 \cdot (v[2] + v[3]) \\ 02 \cdot (v[3] + v[0]) \end{pmatrix} \quad (4)$$

Obviously, a multiplication by the constant 01 is nibble-separable, hence the $0-1$ matrix multiplication in the right side is nibble-separable. Further, let us focus on the multiplication by 02 in the finite field used in MixColumn. Let x and y be two elements of the finite field such that $y = 02 \cdot x$. In a bitwise expression, the relation between x and y is:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} x_7 \\ x_0 + x_7 \\ x_1 \\ x_2 + x_7 \\ x_3 + x_7 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix}. \quad (5)$$

Thus it follows that $y^L = y_3||y_2||y_1||y_0$ can be computed from $x^L = x_3||x_2||x_1||x_0$ and $MSB(x^H) = x_7$. Similarly, y^H can be determined from x^H and $MSB(x^L) = x_4$. Now let us focus on the whole MixColumn operation. Let $w = MixColumn(v)$, where $\{w[i]\}$ are 8-bit bytes and $w = (w[0], w[1], w[2], w[3])$ is a vector. Let $w^H = (w[0]^H, w[1]^H, w[2]^H, w[3]^H)$ and $w^L = (w[0]^L, w[1]^L, w[2]^L, w[3]^L)$ be the vectors of higher and lower nibbles of the vector of bytes. From (4), it follows that w^L can be determined by v^L and the four MSBs: $MSB((v[0] + v[1])^H)$, $MSB((v[1] + v[2])^H)$, $MSB((v[2] + v[3])^H)$ and $MSB((v[3] + v[0])^H)$.

More precisely, if the four MSBs are known, we have a linear bijection between w^L and v^L. Then the 4-bit value of MSB^H required in the computation of the lower nibbles can be defined as follows:

$$MSB^H = \begin{pmatrix} MSB_0^H \\ MSB_1^H \\ MSB_2^H \\ MSB_3^H \end{pmatrix} = \begin{pmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,1 \\ 1\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} MSB(v[0]^H) \\ MSB(v[1]^H) \\ MSB(v[2]^H) \\ MSB(v[3]^H) \end{pmatrix} = M \cdot \begin{pmatrix} v[0]_7 \\ v[1]_7 \\ v[2]_7 \\ v[3]_7 \end{pmatrix} \quad (6)$$

Though there are four free bits $v[i]_7, i \in \{0, 1, 2, 3\}$ at the input, MSB^H has only 2^3 possible values. This comes from the fact that the rank of the binary matrix M in (6) is only 3, which results in a range space of size 2^3. Thus the four bits of MSB^H are linearly dependent and each bit can be computed from the other three. Similar property holds too for the higher nibbles w^H and v^H. $\square$

We can conclude that in order to perform the computing of the higher/lower nibbles through MixColumn, we only need the values of 4 bits from the other half with 2^3 possible combinations. Without the knowledge of the 3 bits, 13 bits of the output can still be determined. This fact will be used further in the partial matching technique introduced in section 3.4.

The parallel-cut division of `Klein` is as follows.

Lemma 1. *The r-round 64-bit cipher* $\texttt{Klein}_r(P, K)$ *with k-bit key K can be split into two r-round 32-bit sub-ciphers* $\texttt{Klein}_r^H(P^H, K^H, msb^L)$ *and* $\texttt{Klein}_r^L(P^L, K^L, msb^H)$ *with $\frac{k}{2}$-bit keys, where msb^L and msb^H are both $6r$-bit values. Then the higher nibbles C^H and the lower nibbles C^L of the ciphertext C can be computed as*

$$C^H = \texttt{Klein}_r^H(P^H, K^H, msb^L),$$
$$C^L = \texttt{Klein}_r^L(P^L, K^L, msb^H).$$

Proof. It follows that the key schedule is nibble-separable, thus all the higher/lower nibbles of all the subkeys can be determined from K^H/K^L. Similarly, all the transformations besides MixNibbles, are nibble-separable as well. In one application of MixNibbles, there are two MixColumns. According to observation 1, the computation of the higher/lower nibbles of one MixColumn requires 3-bit extra information of the MSBs from the lower/higher nibbles, and therefore the whole MixNibbles requires 6 bits. Thus, for r-round `Klein`, the size of the required additional information msb^L/msb^H is $6r$ bits. $\square$

Using the notations in section 2, we know the parallel-cut division of `Klein` has parameters of

$$a_1 = a_2 = 6, b_1 = b_2 = 0, n = 64 \textit{ and } k \in \{64, 80, 96\}.$$

3.4 Attack Algorithm and Its Improvements

Once we know the parameters of the division, the generic parallel-cut MITM attack introduced in section 2 can be applied to `Klein`. The attack algorithms are the same for all versions of `Klein`. The outline of the attack is shown in Fig. 4. According to the formula of the complexity, attacking r-round `Klein` (r is even) with k-bit key requires no more than $\lceil \frac{k}{64} \rceil$ known plaintext-ciphertext pair(s), $2^{\frac{k}{2}+3r} + 2^{\frac{k}{2}+6r-32} + 2^{k-64}$ time and $2^{3r} + 2^{6r+\frac{k}{2}-32}$ memory.

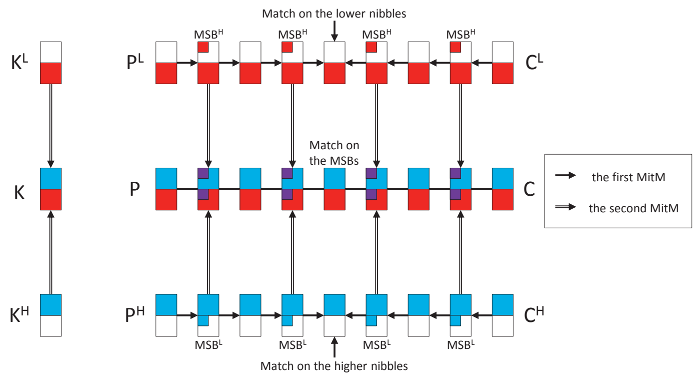

Fig. 4. Outline of the attack on `Klein`

Partial matching technique. In the attack on even number of rounds, we compute the lower/higher nibbles of the middle state in both directions as the matching point. In fact, we could skip the guessing of MSBs in one round and apply the technique of partial matching. It is based on the fact that even if the MSB values are unknown, we could still determine certain bits at the output of the MixNibbles operation.

For the lower nibbles in (5), i.e. the case of multiplication by 02, if the values of (x_0, x_1, x_2, x_3) are known, we could determine the value of the following three bits of y:

$$(y_1 + y_0, y_2, y_3 + y_0) = (x_0, x_1, x_2).$$

When $w = MixColumn(v)$ and the values of $v^L = (v[0]^L, v[1]^L, v[2]^L, v[3]^L)$ are known, we could still determine the value of 3 bits $(w[i]_1 + w[i]_0, w[i]_2, w[i]_3 + w[i]_0)$ for each byte of $w[i]$, where $i \in \{0, 1, 2, 3\}$, or in total $3 \cdot 4 = 12$ bits.

Due to the linear relation of the MSBs of v^H, there is another bit that can be determined too. Consider the values of the least significant bits in w^L, i.e. $(w[0]_0, w[1]_0, w[2]_0, w[3]_0)$. They can be expressed using the bits of v as follows:

$$\begin{pmatrix} w[0]_0 \\ w[1]_0 \\ w[2]_0 \\ w[3]_0 \end{pmatrix} = \begin{pmatrix} v[1]_0 + v[2]_0 + v[3]_0 + v[0]_7 + v[1]_7 \\ v[2]_0 + v[3]_0 + v[0]_0 + v[1]_7 + v[2]_7 \\ v[3]_0 + v[0]_0 + v[1]_0 + v[2]_7 + v[3]_7 \\ v[0]_0 + v[1]_0 + v[2]_0 + v[3]_7 + v[0]_7 \end{pmatrix}. \quad (7)$$

Therefore the value $w[0]_0 + w[1]_0 + w[2]_0 + w[3]_0 = v[0]_0 + v[1]_0 + v[2]_0 + v[3]_0$ can be determined without the knowledge of v^H. Similarly, for the higher nibbles, the equation takes the form:

$$\begin{pmatrix} w[0]_4 \\ w[1]_4 \\ w[2]_4 \\ w[3]_4 \end{pmatrix} = \begin{pmatrix} v[1]_4 + v[2]_4 + v[3]_4 + v[0]_3 + v[1]_3 + v[0]_7 + v[1]_7 \\ v[2]_4 + v[3]_4 + v[0]_4 + v[1]_3 + v[2]_3 + v[1]_7 + v[2]_7 \\ v[3]_4 + v[0]_4 + v[1]_4 + v[2]_3 + v[3]_3 + v[2]_7 + v[3]_7 \\ v[0]_4 + v[1]_4 + v[2]_4 + v[3]_3 + v[0]_3 + v[3]_7 + v[0]_7 \end{pmatrix}. \quad (8)$$

Again $w[0]_4 + w[1]_4 + w[2]_4 + w[3]_4 = v[0]_4 + v[1]_4 + v[2]_4 + v[3]_4$, thus it can be determined without the knowledge of v^L.

As a result, we can determine the values of $3 \times 4 + 1 = 13$ bits of w^L/w^H only from the value of lower/higher nibbles of the input v^L/v^H.

Attack on odd rounds. Using the above partial matching technique, we can skip one round in both of the first two MITMs (recall that PC MITM uses three distinct MITMs). Therefore, the computational complexity of one direction can be reduced by a factor of 2^6 and the size of the matching point is also reduced from 32 bits to $13 \times 2 = 26$ bits (2^{13} bits for one MixColumn). The expected number of matches $2^{6r-6-26} = 2^{6r-32}$ in the MITMs is not affected since the size of matching sets and the size of the filter are both reduced by a factor of 2^6.

The complexity of the attack on even number of rounds cannot be reduced using this technique, as we are able to apply the reduction only in one round and, more importantly, in one direction of the MITM – the second would remain the same thus the combined complexity would stay intact. On the other hand, for the attack on odd number r of rounds, the complexity can be reduced. We split the cipher with $\frac{r-1}{2}$ rounds at the left side, and $\frac{r-1}{2}$ rounds at the right side, with an additional single round in the middle. This is precisely the round where the partial matching technique is applied. Thus the complexity of MITM becomes $2^{6\frac{r-1}{2}} = 2^{3r-3}$ and the number of remaining candidates is still 2^{6r-32}.

The complexity of the attack on even number of rounds, per half key K^L, is 2^{3r}. Hence a unified expression for the complexity on both even and odd r would be $2^{6\lfloor\frac{r}{2}\rfloor}$. As a result, the total complexity of our attack on r-round `Klein` becomes $2^{6\lfloor\frac{r}{2}\rfloor+\frac{k}{2}} + 2^{6r+\frac{k}{2}-32} + 2^{k-64}$ computations and $2^{6\lfloor\frac{r}{2}\rfloor} + 2^{6r+\frac{k}{2}-32}$ memory.

A time-memory trade-off. Further we show that the time and memory requirements of our attacks are flexible to a certain extend, i.e. we can reduce the memory requirement while increasing the time complexity of the first two MITMs only. The idea is to choose and fix the MSBs of the nibbles in the first t rounds. Since we need 12 bit guesses per round, the whole attack (the three MITMs) will be repeated 2^{12t} times to find all the filtered keys.

Before we choose the values of the MSBs, a precomputation is needed. For all $2^{\frac{k}{2}}$ K^L/K^H, we compute the value of the MSBs in round $t+1$ using the guessed MSBs in round 1 to round t and build a lookup table. Once a $12t$-bit value of the MSBs is fixed, we can directly find about $2^{\frac{k}{2}-6t}$ corresponding half key by indexing the $6t$-bit MSB value in round $t+1$.

For a chosen value of the MSBs in the first t rounds and all the $2^{\frac{k}{2}-6t}$ corresponding keys, the first two MITMs are now on $r-t$ rounds, thus require $2^{\frac{k}{2}+6\lfloor\frac{r-t}{2}\rfloor-6t}$ time and $2^{6\lfloor\frac{r-t}{2}\rfloor}$ memory, with $2^{\frac{k}{2}+6r-12t-32}$ candidates remaining. The remaining third MITM is only on $12(r-t)$ bits, and requires $2^{\frac{k}{2}+6r-12t-32}$ time and memory, with $2^{k-12t-64}$ expected matches.

For all values of the MSBs in t rounds, the above steps are repeated 2^{12t} times. Therefore, the total complexity is $T(k,r,t) = 2^{12t} \cdot (2^{\frac{k}{2}+6\lfloor\frac{r-t}{2}\rfloor-6t} +$

Table 2. Practical Result on 4-round Klein-64

Plaintext	0000 0000 0000 0000
Ciphertext	0000 0000 0000 0000
Key	DE7F A226 0917 5484

$2^{\frac{k}{2}+6r-12t-32}+2^{k-12t-64}) = 2^{\frac{k}{2}+6\lfloor\frac{r+t}{2}\rfloor}+2^{\frac{k}{2}+6r-32}+2^{k-64}$ time and $M(k,r,t) = 2^{6\lfloor\frac{r-t}{2}\rfloor} + 2^{6(r-t)+\frac{k}{2}-32}$ memory. Note, when $t = 0$, one obtains the same complexities as mentioned in the previous sections.

3.5 Final Results

The round functions for all three versions of Klein are identical, thus the technique from Section 3.4 can be applied to all of them. In the formulae for the complexities, when the parameter t changes, the time and memory only for the first level MITMs (i.e. the MITMs on the lower/higher halves) are affected. When t increases, the first level MITMs require more time and less memory. In order to find the best attacks in terms of maximal number of rounds, first we find the largest attackable number of rounds with $t = 0$. Then we compare the computational costs of the first and second level MITMs. If the first level MITM requires more time complexity than the second level MITM, then the memory cannot be reduced while maintaining the same number of rounds – otherwise the time complexity would increase, making the attack worse than the simple exhaustive key search. However, when the second level MITM is more costly, then we try to find the value for the parameter t to reduce the memory requirement of our attack. The final results are as follows.

For Klein-64, we can attack 10 out of 12 rounds. The parameters for the attack are $k = 64, r = 10$ and $t = 0$. The complexity is $T(64, 10, 0) = 2^{62}$ time and $M(64, 10, 0) = 2^{60}$ memory.

For Klein-80, we can attack 11 out of 16 rounds. The optimal parameters for the attack are $k = 80, r = 11$ and $t = 0$. The complexity is $T(80, 11, 0) = 2^{74}$ time and $M(80, 11, 0) = 2^{74}$ memory.

For Klein-96, we can attack 13 out of 20 rounds and apply the time-memory tradeoff to reduce the memory requirement without increasing the time complexity. This comes from the fact that our time complexity increases only in the first two MITMs, but not in the third. Thus the optimal parameters for the attack are $k = 96, r = 13$ and $t = 2$. The complexity is $T(96, 13, 2) = 2^{94}$ time and $M(96, 13, 2) = 2^{82}$ memory.

In order to confirm the correctness of our approach, we have implemented the attack on 4-round Klein-64 - it takes 2^{44} computations and 2^{24} memory. We have set the plaintext and ciphertext to all-zero vectors and successfully recovered the 64-bit key. The result is given in Table 2.

4 Conclusion

We have proposed a new type of meet-in-the-middle attack which splits the cipher along the execution flow in a way such that the produced subciphers have as less as possible exchanging bits. We have shown that `Klein` can be divided into two smaller ciphers with only 6 exchanged bits per round. Then, two MITM attacks have been used to attack the two independent parts, and one final MITM to connect them. This results in attacks on 10 rounds for `Klein`-64, 11 rounds for `Klein`-80, and 13 rounds for `Klein`-96. The main advantage of the attack is the data requirement – it is completely practical. As our divide-and-conquer approach is based on the MITM attack, the data complexity is minimal – to recover the key we require only one or two pairs of known plaintext-ciphertexts.

The three-step (three MITMs) approach we use in the PC MITM is our way of taking advantage of the exchanged bits produced after the parallel-cut division. However, by no means it should be considered that this is the only way and some MITMs can be avoided. In particular, when the number of exchanged bits is relatively small, and when the target is to find preimages for hash functions (rather than recover the key), it might be beneficial first to fix the exchanged bits, and then to search preimages for the smaller hash functions. Also, it is not required to divide the initial cipher into two subciphers that necessary have the same state/key sizes. Rather, the division should be such that the number of exchanged bits between the ciphers is equal. This results in balanced complexity of the first two MITMs.

We would like to point out two open problems, solution of which could lead to improvements of the complexity of PC MITM. The first is related to the memory requirement of our approach. Though MITM attacks in general have a memoryless variant, in the case (as ours) when one needs to find not one but *all* the matches, the proposed approach from [7] fails. Intuitively, it seems such memoryless MITM for n-bit function cannot exist when the time complexity is bounded by 2^n – otherwise one has to be sure that in the Floyd cycle finding algorithm, the starting vertex for each separate collision is at different path that leads to a cycle. A formal proof, either confirming or disapproving our conjecture, would contribute significantly to the field of cryptography. The second open problem is related to the way we perform the PC MITM attack – first we match separately in the middle of the lower/higher nibble states, and then we match on the guessed bits. In general this can be seen as a problem of finding *partial pair collisions between four functions* – each with two inputs and two outputs. The functions are coupled in pairs, and each pair has one general independent input and one input coming from a function of the other pair. The problem consists in finding the two independent inputs that produce collisions at the second output of each pair. We have solved the problem by guessing the exchanged inputs between the pairs, generating all independent inputs, and finally matching the guesses. A better solution in terms of time/memory complexity, exploiting simultaneously the dependency between the inputs and outputs, might exist.

We argue that the PC MITM is not specific only to `Klein` and it can be applied not only to ciphers that allow nibble separation. The area of application is much wider and can include word-oriented primitives as well. A good starting point are the block ciphers TEA [9] and XTEA [8] – two primitives that operate with 32-bit words and allow parallel-cut division with a relatively small number of exchanged bits per round achieved with the shift operations.

References

1. Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.
2. Jean-Philippe Aumasson, María Naya-Plasencia, and Markku-Juhani Saarinen. Practical Attack on 8 Rounds of the Lightweight Block Cipher KLEIN. In Daniel Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011*, volume 7107 of *Lecture Notes in Computer Science*, pages 134–145. Springer Berlin / Heidelberg, 2011.
3. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matt Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 199–214. Springer Berlin / Heidelberg, 2004.
4. Whitfield Diffie and Martin E Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.
5. Zheng Gong, Svetla Nikova, and Yee Law. KLEIN: A New Family of Lightweight Block Ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2012.
6. Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of full KLEIN-64. In *FSE Rump Session*, 2013.
7. Hikaru Morita, Kazuo Ohta, and Shoji Miyaguchi. A switching closure test to analyze cryptosystems. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 183–193. Springer Berlin / Heidelberg, 1992.
8. Roger M. Needham and David J. Wheeler. TEA extensions. Technical report, University of Cambridge, 1997.
9. David J. Wheeler and Roger M. Needham. TEA, a tiny encryption algorithm. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.
10. Xiaoli Yu, Wenling Wu, Yanjun Li, and Lei Zhang. Cryptanalysis of reduced-round KLEIN block cipher. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 7537 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2011.