Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64

Bo Zhu and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
{bo.zhu,ggong}@uwaterloo.ca

Abstract. This paper investigates a new approach to analyze symmetric ciphers by guessing intermediate states and dividing algorithms to consecutive sub-ciphers. It is suitable for ciphers with simple key schedules and block sizes smaller than key lengths. A thorough theoretical analysis of this multidimensional method is given, and new attacks on the block cipher family KATAN are proposed by applying this method, which can attack 175-round KATAN32, 130-round KATAN48 and 112-round KATAN64 faster than exhaustive key search.

Keywords: Multidimensional, meet-in-the-middle, cryptanalysis, KATAN

1 Introduction

Lightweight devices such as RFID chips and wireless sensor networks have become popular these days, because such tiny devices bring more convenience to people's daily lives and are able to solve a large number of traditional problems at very low costs. Security and privacy protections for such devices are therefore highly demanded. However, original designs of cryptographic algorithms and protocols, such as TLS/SSL, can hardly be used under such circumstances due to limited computational and storage capacities.

Recently, many new cryptographic designs for lightweight devices have been carried out. For example, the KATAN/KTANTAN families of block ciphers [1] and the authenticated encryption algorithm Hummingbird-2 [2] are devised specifically for constrained environments. The block cipher PRINTcipher [3] is designed to be compact enough for integrated circuit printing. A 64-bit version block cipher, LED [4], is proposed based on the structure of AES, which has similar security evaluation but smaller implementation footprints. Security evaluation of these lightweight algorithms becomes a very important work for researchers.

Meet-in-the-middle (MITM, hereafter) attack was first introduced by Diffie and Hellman in [5] for cryptanalysis of DES, and is a generic method to analyze high-level structures of cryptographic algorithms. Its fundamental idea is that if the target algorithm can be decomposed into two consecutive parts and the computation of each part only involves portions of master keys, then we can investigate the security level of each part separately and finally check the consistence of the results from both sides. Since evaluating two smaller segments usually requires much less work, the overall time complexity to analyze the complete algorithm could decrease dramatically.

Inspired by the recent development of MITM techniques for cryptanalysis, such as biclique attacks [6, 7] which create the first single-key attacks on full AES and IDEA, and splice-and-cut attacks for computing pre-images of MD5 [8], SHA-0 and SHA-1 [9], we investigated a new method in depth, i.e. ciphers are first divided into consecutive sub-ciphers by guessing certain intermediate states, then MITM attacks are applied to these sub-ciphers separately, and results are finally brought together to eliminate wrong keys. We applied this multidimensional approach to the block cipher family KATAN, and found the best cryptanalysis results so far. The papers [10, 11] have the basic idea about MITM attacks using one guess, but those attacks only succeed in improving memory and data complexities, but not time complexity, of the previous work [12]. In addition, our multidimensional MITM attack

can be seen as a generalized framework for the attacks in [10, 11]. Here we also note that there is a independent work [13] with similar ideas, but its authors focus on optimizing time-memory trade-offs for composite problems, and their analysis is only applied to the cases in which all sub-ciphers have independent keys.

The comparisons of our new cryptanalysis results on KATAN with existing ones are summarized in Table 1. The notation KP in the table stands for *known plaintexts*.

Table 1. Comparisons of Previous and New Cryptanalysis Results on Reduced-round KATAN.

Cipher	Number of Rounds	Time Complexity	Memory Complexity	Data Complexity	Reference
KATAN32	115	2^{79}	Not Given	2^{32} KP	[14]
	175	$2^{79.30}$	$2^{79.58}$	3 KP	Sec. 3.3
KATAN48	100	2^{78}	2^{78}	128 KP	[15]
	130	$2^{79.45}$	$2^{79.00}$	2 KP	Sec. 3.4
KATAN64	94	$2^{77.68}$	$2^{77.68}$	116 KP	[15]
	112	$2^{79.45}$	$2^{79.00}$	2 KP	Sec. 3.4

2 Multidimensional Meet-in-the-Middle Attack

This section will first briefly introduce original MITM attacks, and then discuss how to extend them to multidimensional cases.

2.1 Meet-in-the-Middle Attack

We take Double-DES (2DES) to explain the idea of MITM attacks. Use $c = DES_k(p)$ to denote one DES encryption, where k is the 56-bit master key, and p and c are the plaintext and ciphertext. 2DES uses two different keys k_1 and k_2, and its encryption is computed as

$$c = 2DES_{(k_1,k_2)}(p) = DES_{k_2}(DES_{k_1}(p)) \ .$$

The total number of key bits is equal to $2 \cdot 56 = 112$, so the time complexity of exhaustive key search for 2DES is 2^{112}. Using the MITM method, we can first compute $v = DES_{k_1}(p)$ for all possible k_1's, and store all v's into a set S with corresponding k_1's. The time complexity of this step is dominated by 2^{56}. Secondly, from the ciphertext side, we can compute $v' = DES^{-1}_{k_2}(c)$ for each possible k_2, and then check whether v' is in the set S. If we find a match, then the corresponding key pair (k_1, k_2) is possibly the right one. In this way, we only need to calculate DES for $2 \cdot 2^{56} = 2^{57}$ times, which is much less than the expected 2^{112}. This is the reason why we should use Triple-DES, rather than Double-DES, to obtain a reasonably large security margin.

Suppose a cipher $c = E(k, p)$ can be decomposed into two consecutive sub-ciphers $E_f(k_f, \cdot)$ and $E_b(k_b, \cdot)$, i.e. $c = E_b(k_b, E_f(k_f, p))$, where k_f and k_b are the sub-keys used in E_f and E_b, as shown in Fig. 1. Here f and b are the abbreviations for *forward* and *backward*.

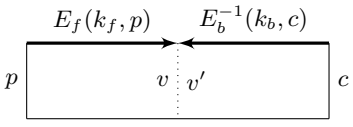

Fig. 1. An illustration of meet-in-the-middle attacks.

The steps of MITM attacks can be written as follows.

- MITM phase:
  1. Compute every possible $v = E_f(k_f, p)$ by iterating possible k_f. Collect v's into a set S.
  2. For each possible k_b, compute $v' = E_b^{-1}(k_b, c)$. Check whether $v' \in S$. If so, output the corresponding key pair (k_f, k_b) as a possible correct key.
- Brute-force testing phase: If the MITM phase outputs more than one pair of (k_f, k_b), then we need to use additional plaintext-ciphertext pairs to perform complete encryptions to test them and find the correct pair.

Let us use $|\cdot|$ to represent the bit-length of a variable, and n to denote the block size of a cipher, e.g., $n = |p| = |c|$. For simplicity, we assume bit-lengths of intermediate states are smaller than block sizes, e.g., $|v| \leq n$, in the following content. The time complexity for Step 1 of the MITM phase is $2^{|k_f|}$, and for Step 2 it is $2^{|k_b|}$. During the MITM phase, wrong keys have the probability of $1/2^{|v|}$ to obtain a false positive, so if k_f and k_b do not have common key bits, the number of wrong keys passing the MITM phase will be $2^{|k_f|+|k_b|}/2^{|v|} = 2^{|k_f|+|k_b|-|v|}$. Let k_c consist of all the key bits contained in both k_f and k_b, the number of remaining keys will be $2^{|k_f|+|k_b|-|k_c|-|v|}$. We further assume k is the master key which consists of all the key bits of k_f and k_b, and $|v|$ is equal to the block size n, then we have

$$2^{|k_f|+|k_b|-|k_c|-|v|} = 2^{|k|-|v|} = 2^{|k|-n}.$$

This can be easily understood from the information theory's point of view: Since we have information of an n-bit plaintext-ciphertext pair (p, c), we can only reduce the key space to $1/2^n$ of the original. This is also the reason why if $|v| > n$ we cannot filter out more wrong keys[1]. After this MITM phase, we can simply deploy brute-force testing to remove rest wrong keys.

The time complexity of the first attempt of brute-force testing will be equal to $2^{|k|-n}$. The probability of wrong keys passing the testing is $1/2^n$ on average, so $2^{|k|-2n}$ keys will pass the first testing. If $2^{|k|-2n}$ is still larger than 1, we can use another pair of (p, c) to perform additional testing to further reduce the key space. The overall time complexity of the brute-force testing phase will be $2^{|k|-n} + 2^{|k|-2n} + 2^{|k|-3n} + \cdots$, and this phase needs $\lceil (|k| - n)/n \rceil$ pairs of plaintexts and ciphertexts. To sum up, the total time complexity is

$$2^{|k_f|} + 2^{|k_b|} + 2^{|k|-n} + 2^{|k|-2n} + 2^{|k|-3n} + \cdots \approx 2^{|k_f|} + 2^{|k_b|} + 2^{|k|-n},$$

and the total data complexity is $\lceil (|k| - n)/n \rceil + 1 = \lceil |k|/n \rceil$. Similar analysis can be found in [16].

When a matching key pair (k_f, k_b) is found, it can be tested instantly, so we do not need to save it in memory and wait for other candidate keys. Therefore, the major memory consumption of this attack comes from constructing the set S. There are many kinds of data structures to produce S, such as hash tables. Actually, constructions and look-up algorithms of S also have influence on the overall attack time. The look-up time is generally omitted since it is usually much less than a complete encryption. We suggest to use tables whose indices are (parts of) matching values, e.g., v in the above example, and let each entry in the tables point to a linked list of corresponding sub-keys. Despite of different constructions of S, memory complexities should be $2^{|k_f|}$ at least.

2.2 Multidimensional Meet-in-the-Middle Approach

When designing block ciphers for environment-constrained devices, we usually prefer to adopt small block sizes for efficient performance. However, due to security requirements, master keys cannot be too short. This usually leads us to cipher designs with key sizes larger than block sizes. Although this is perfectly valid, it makes us possible to guess certain short intermediate states of ciphers and divide them into small sub-ciphers for easier analysis.

Suppose we first guess an intermediate state g, and perform two MITM attacks on the sub-ciphers divided by g, as shown in Fig. 2. In the simplest case, assuming the sub-keys k_{f_1}, k_{b_1}, k_{f_2} and k_{b_2} do not have common key bits, the steps of the attack can be described as follows.

[1] Nevertheless, if $|v| < n$ we will get a larger resultant key space.

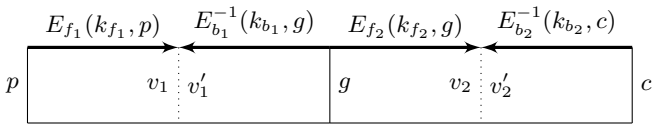

Fig. 2. Meet-in-the-middle attacks with one guess.

1. Compute $v_1 = E_{f_1}(k_{f_1}, p)$ for each possible k_{f_1}, and put all k_{f_1}'s into a table T_1 indexed by v_1, each entry of which is a set of certain k_{f_1}'s.
2. Compute $v_2' = E_{b_2}^{-1}(k_{b_2}, c)$ for each possible k_{b_2}, and put all k_{b_2}'s to a table T_2' (similar as T_1) indexed by v_2'.
3. For each possible guess of g:
   (a) Compute $v_1' = E_{b_1}^{-1}(k_{b_1}, g)$ for each possible k_{b_1}, and maintain a table T_1' of k_{b_1} indexed by v_1'.
   (b) Compute $v_2 = E_{f_2}(k_{f_2}, g)$ for each possible k_{f_2}, and maintain a table T_2 of k_{f_2} indexed by v_2.
   (c) Every matching pair (k_{f_1}, k_{b_1}) for $v_1 = v_1'$, together with each matching pair (k_{f_2}, k_{b_2}) for $v_2 = v_2'$, forms a candidate key for the whole encryption. We use additional plaintext-ciphertext pairs to perform brute-force testing on these candidate keys. If one key passes all tests, then output it as the correct key.

The time complexity of this attack without the brute-force testing part is

$$2^{|k_{f_1}|} + 2^{|k_{b_2}|} + 2^{|g|} \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|}). \tag{1}$$

We only need to recompute $E_{b_1}^{-1}$ and E_{f_2} for different g's, but not E_{f_1} and $E_{b_2}^{-1}$. For each guessed value of g, the MITM attack from g to c will reduce the size of the key space to $2^{|k|-|v_2|}$ and the second attack for the interval p to g will further reduce it to $2^{|k|-|v_1|-|v_2|}$, so after the two MITM attacks the total number of keys left is $2^{|g|} \cdot (2^{|k|-|v_1|-|v_2|}) = 2^{|k|+|g|-|v_1|-|v_2|}$. Assuming $|g| = |v_1| = |v_2| = n$, we will have $2^{|k|+|g|-|v_1|-|v_2|} = 2^{|k|-n}$, which is consistent with the analysis for original MITM attacks in the last subsection. The total time complexity of the brute-force step is still $2^{|k|-n} + 2^{|k|-2n} + \cdots$.

The memory complexity of the attack is $2^{|k_{f_1}|} + 2^{|k_{b_1}|} + 2^{|k_{f_2}|} + 2^{|k_{b_2}|}$, since we need to store T_1, T_1', T_2 and T_2' in memory. The data complexity of the attack is still $\lceil |k|/n \rceil$.

In general cases, the sub-keys, k_{f_1}, k_{b_1}, k_{f_2} and k_{b_2}, would involve many common key bits, so the above attack cannot be applied directly. A straightforward way to solve this is to treat every sub-key bit as an independent new variable. This technique has actually been used in other cryptanalysis methods such as [15], and have improved cryptanalysis results on several block ciphers. But by carefully investigating ciphers' detailed design we may get better results. For example, we can perform linear transformations before matching sub-keys, or study round functions to perform partial encryptions or decryptions. We will show real attack examples on the block cipher family KATAN32/48/64 in the next section.

Certainly, we can guess more intermediate states and segment ciphers into smaller pieces, in which case we may be able to find better attacks. MITM attacks with multiple guesses are illustrated in Fig. 3. For simplicity of description, hereafter we denote the MITM attacks with multiple guesses as multidimensional MITM (MD-MITM) attacks, and especially the attacks with n sub-ciphers will be nD-MITM. For example, the above attack with one guess is a 2D-MITM attack, and original MITM attacks can be seen as 1D-MITM attacks.

The steps of an $(i+1)$D-MITM attack can be briefly stated as follows.

1. Construct a table T_1 of k_{f_1} by computing $v_1 = E_{f_1}(k_{f_1}, p)$.
2. Construct a table T_{i+1}' of $k_{b_{i+1}}$ by computing $v_{i+1}' = E_{b_{i+1}}^{-1}(k_{b_{i+1}}, c)$.
3. For each guess of g_1:
   (a) Construct a table T_1' by computing $v_1' = E_{b_1}^{-1}(k_{b_1}, g_1)$ which is to match with T_1.
   (b) Construct a table T_2 by computing $v_2 = E_{f_2}(k_{f_2}, g_1)$.

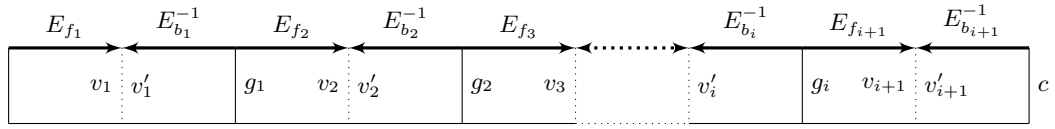

Fig. 3. General process of multidimensional meet-in-the-middle attacks with multiple guesses.

(c) For each guess of g_2:
  i. Construct a table T_2' by computing $v_2' = E_{b_2}^{-1}(k_{b_2}, g_2)$ which is to match with T_2.
  ii. $\cdots$ (Perform recursive operations till g_i.)
  iii. For each guess of g_i:
    A. Construct a table T_i' by computing $v_i' = E_{b_i}^{-1}(k_{b_i}, g_i)$ to match with T_i.
    B. Compute $v_{i+1} = E_{f_{i+1}}(k_{f_{i+1}}, g_i)$, which can form a table T_{i+1} to match with T_{i+1}'
    C. Perform brute-force testing on each combination of matching sub-key pairs from (T_1, T_1'), (T_2, T_2'), $\cdots$, (T_{i+1}, T_{i+1}'), and output the passing key as the correct one.

The time complexity of the phase of MITM attacks with multiple guesses is given as

$$\begin{aligned} &2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|} + 2^{|g_1|} \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|} + 2^{|g_2|} \cdot (2^{|k_{b_2}|} + 2^{|k_{f_3}|} + \cdots)) \\ &= 2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|} + 2^n \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|}) + 2^{2n} \cdot (2^{|k_{b_2}|} + 2^{|k_{f_3}|}) + \cdots , \end{aligned} \tag{2}$$

if we assume $|g_1| = |g_2| = \cdots = n$. Please note that the order of $g_1, g_2, \cdots, g_i$ does not matter, and we can first guess any ones of them, in which case we may find another order for a better attack.

We only use one known plaintext-ciphertext pair before brute-force testing in the above MD-MITM attack, so it is easy to see MD-MITM attacks have the same data complexities as 1D-MITM attacks.

The memory complexities of MD-MITM attacks are upper bounded by the sum of memory consumptions of $T_1, T_1', T_2, \cdots, T_{i+1}'$. To get the minimum value for the time complexity equation (2), the sizes of $T_1', T_2, \cdots, T_{i+1}$, i.e. $2^{|k_{b_1}|}, 2^{|k_{f_2}|}, \cdots, 2^{|k_{f_{n+1}}|}$, should be much smaller than the sizes of T_1 and T_{i+1}', i.e. $2^{|k_{f_1}|}$ and $2^{|k_{b_{i+1}}|}$. So it is safe to ignore $T_1', T_2, \cdots, T_{i+1}$ here and give an upper bound for the memory complexities of these simple MD-MITM attacks as $2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|}$.

3 Application to KATAN

In this section, we first briefly describe the KATAN family of block ciphers, and present our cryptanalysis results on KATAN by using MD-MITM attacks.

3.1 KATAN Family of Block Ciphers

KATAN [1] is a block cipher family designed for constrained devices and embedded systems, which has a very small footprint and an acceptable security level. It consists of three versions with different block sizes, 32, 48 and 64 bits, which are named KATAN32, KATAN48 and KATAN64 respectively. Despite of their different block sizes, they all use 80-bit master keys. The structure of KATAN is shown in Fig. 4.

In the encryption process of KATANn, the plaintext p is first divided to two pieces and loaded into the registers L_1 and L_2. Next, Two nonlinear functions defined by the equations (3) are operated on L_1 and L_2 respectively.

$$\begin{aligned} f_a[L_1] &= L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a \\ f_b[L_2] &= L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b \end{aligned} \tag{3}$$

In the above equations, x_i and y_j are predefined indices for different versions of KATAN, and IR is an irregular update sequence to prevent self-similarity attacks. The parameters x_i, y_j are IR are given in

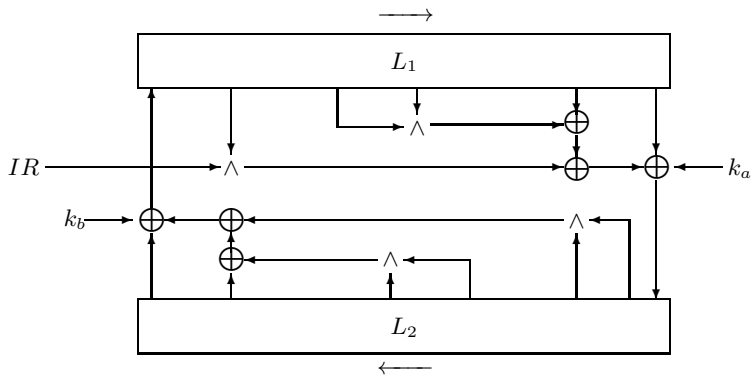

Fig. 4. The structure of KATAN [1].

the appendix. k_a and k_b are two sub-key bits produced from the 80-bit master key by a linear feedback shift register (LFSR). The 80-bit master key K is loaded as the initial state of the LFSR, and each output bit is used as a sub-key bit sequentially. Assuming $\{k_i\}$ is the output sequence of the LFSR, k_i is equal to the i-th bit of the master key if $i < 80$, and $k_i = k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13}$ if $i \geq 80$. For r-th round, $k_a = k_{2r-2}$ and $k_b = k_{2r-1}$, where $1 \leq r \leq 254$.

For KATAN32, after computing $f_a[L_1]$ and $f_b[L_2]$, the registers L_1 and L_2 are shifted to left one time, and the most significant bits of L_1 and L_2 are discarded. Next, $f_a[L_1]$ is fed into the least significant bit of L_2, and $f_b[L_2]$ is put into L_1. After 254 rounds of such updating operations, the states of L_1 and L_2 are concatenated as the ciphertext c.

KATAN48 and KATAN64 have the same structure as KATAN32, and have the same total number of rounds, i.e. 254, but in each round L_1 and L_2 of KATAN48 and KATAN64 are updated two and three times respectively, by using the same k_a and k_b.

3.2 New Attacks on KATAN32

Use s_i to denote the 32-bit intermediate state after partial encryption of i rounds, which implies $s_0 = p$ and $s_{254} = c$. Let us show a simplest case of 2D-MITM on KATAN32 first and improve it in following discussions. For simplicity, we use $E_i(s)$ to denote $E_{f_i}(k_{f_i}, s)$, and $D_j(s)$ to denote $E^{-1}_{b_j}(k_{b_j}, s)$. $k_{i...j}$ is the sub-key containing all the sub-key bits whose indices are from i to j. The detailed attack procedure is as follows.

1. Compute $s_{40} = E_1(s_0)$ using every possible $k_{f_1} = k_{0...79}$, and compute $k_{80...127}$ by using linear functions derived from the LFSR. Put each $k_{0...79}$ in a table T_1 indexed by s_{40} and $k_{80...127}$. Each entry of the table should have one element on average.
2. Compute $s'_{88} = D_2(s_{128})$ using every possible $k_{b_2} = k_{176...255}$, and compute $k_{128...175}$ by using linear functions derived from the LFSR. Store each $k_{176...255}$ in a table T'_2 indexed by s'_{88} and $k_{128...175}$. Similarly each entry of the table has one element on average.
3. For each guess of $g = s_{64}$.
   (a) Compute $s'_{40} = D_1(s_{64})$ for each $k_{b_1} = k_{80...127}$, and then find the matching key $k_{0...77}$ in T_1. On average there is only one such key, and put it in a set S.
   (b) Compute $s_{88} = E_2(s_{64})$ for each $k_{f_2} = k_{128...175}$, and similarly find the matching key $k_{176...255}$ in T'_2. Then compute $k_{0...79}$ by using the linear relations defined by the LFSR, and check whether it is also in the set S. If so, perform brute-force testing on this candidate key. If it passes all tests, output it as the correct master key.

To fairly compare with the complexities of existing attacks, we adopt the method proposed in [17] to estimate the time complexity of the above attack on KATAN. Use R_{f_1}, R_{b_1}, R_{f_2} and R_{b_2} to denote the numbers of rounds involved in different phase of the 2D-MITM attack, and the total number of attacked rounds is R. The time complexities without brute-force testing are computed as follows.

$$2^{|k_{f_1}|} \cdot \frac{R_{f_1}}{R} + 2^{|k_{b_2}|} \cdot \frac{R_{b_2}}{R} + 2^n \cdot \left(2^{|k_{b_1}|} \cdot \frac{R_{b_1}}{R} + 2^{|k_{f_2}|} \cdot \frac{R_{f_2}}{R} \right)$$

Here we simply ignore the time complexities of linear transformations on sub-keys, because these computations usually only involve several linear operations, which are much more cost-efficient than the iterations of nonlinear partial encryptions or decryptions, and the exhaustive key search also needs to compute the sub-keys on the fly by costing almost equivalent time.

In the above 2D-MITM attack on KATAN32, since $|k_{f_1}| = |k_{b_2}| = 80$ and $|k_{b_2}| = |k_{f_2}| = 48$, its total time complexity is

$$2^{80} \cdot \frac{40}{128} + 2^{80} \cdot \frac{40}{128} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{128} + 2^{48} \cdot \frac{24}{128} \right) + 2^{80-32} \approx 2^{80},$$

where 2^{80-32} is the time complexity of the brute-force testing phase. Please note that 2^{80} is exactly the time complexity of exhaustive key search on KATAN. The memory complexity of the attack is $2^{80} + 2^{80} + 2^{80-32} \approx 2^{81}$, since we need to store T_1, T_2 and S in memory. The data complexity is still the same as 1D-MITM attacks, i.e. $\lceil 80/32 \rceil = 3$.

Reducing Complexities. To make the time and memory complexities of the above attack to be under 2^{80}, we can reduce the numbers of attacked rounds of first forward and second backward phases by one. But in this case, when constructing T_1 (or similarly T_2'), we cannot simply use s_{38} and $k_{b_1} = k_{78\ldots125}$ as indices like in the above attack, because $k_{f_1} = k_{0\ldots77}$ does not have full 80-bit information of the master key, certain bits of k_{b_1} depend on the values of k_{78} and k_{79}. Under such circumstance, we propose the following attack procedure.

1. Compute $s_{39} = E_1(s_0)$ using every possible $k_{f_1} = k_{0\ldots77}$, and compute $k'_{80\ldots125}$ by treating $k_{78} = k_{79} = 0$. Put each $k_{0\ldots77}$ in a table T_1 indexed by s_{39} and $k'_{80\ldots125}$. Each entry of the table should have one element on average.
2. Compute $s'_{87} = D_2(s_{126})$ using every possible $k_{b_2} = k_{174\ldots251}$, and compute $k'_{126\ldots171}$ by treating $k_{172} = k_{173} = 0$. Store each $k_{174\ldots251}$ in a table T_2' indexed by s'_{87} and $k'_{126\ldots171}$. Similarly each entry of the table has one element on average.
3. For each guess of $g = s_{63}$:
   (a) Compute $s'_{39} = D_1(s_{63})$ for each $k_{b_1} = k_{78\ldots125}$, and compute $k'_{80\ldots125}$ by deducting k_{78} and k_{79} from $k_{80\ldots125}$. Then use s'_{39} and $k'_{80\ldots125}$ to find the matching $k_{0\ldots77}$ in T_1. On average there is only one matching $k_{0\ldots77}$, combine it with the k_{78} and k_{79} to form the master key. Put the candidate master key in a set S.
   (b) Compute $s_{87} = E_2(s_{63})$ for each $k_{f_2} = k_{126\ldots173}$, and compute $k'_{126\ldots171}$ by deducting k_{172} and k_{173} from $k_{126\ldots172}$. Then use s_{87} and $k'_{126\ldots171}$ to find the matching $k_{174\ldots251}$ in T_2'. Compute $k_{0\ldots79}$ from $k_{172\ldots251}$ by using the linear relations defined by the LFSR, and check whether it is also in the set S. If so, perform brute-force testing on it.

The overall time complexity of this attack on 126-round KATAN32 is

$$2^{78} \cdot \frac{39}{126} + 2^{78} \cdot \frac{39}{126} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{126} + 2^{48} \cdot \frac{24}{126} \right) + 2^{80-32} \approx 2^{79.10},$$

and its memory complexity is $2^{78} + 2^{78} + 2^{48} \approx 2^{79}$. This attack already reaches the number of rounds more than any previous attack on KATAN32, but we still have room for improvements.

Increasing Attacked Rounds. We can see there is a large time complexity gap between the MD-MITM and brute-force testing phases in the above attacks on KATAN32. The time complexities of brute-force testing are always 2^{48}, and the complexities of MITM phases are close to 2^{80}. If we can balance complexities of the two phases, the overall time complexities may drop. One way to do this is reducing bit-lengths of the intermediate states for matching, i.e. $v_1, v_2, \cdots$. As a result, computing incomplete parts of original $v_1, v_2, \cdots$ may not need to use up all of the sub-key bits, and may extend the attack to more rounds. In this way, more candidate keys are left to be tested in brute-force phases. This technique is called partial matching, and has been used in various papers, such as [16, 17].

By adopting the partial matching technique, we are to extend our attack on 126-round KATAN32 to more number of rounds. For simplicity, we only use partial matching in the second MITM attack. After analysis, we found the best meeting point is s_{87} and the second MITM attack ends at s_{152}. Based on the 78-bit information of $k_{b_2} = k_{226...303}$, we can still compute 2 bits of s_{87}. By using these 2 bits for matching, there will be 2^{78} candidate keys left for brute-force testing, and thus the total time complexity of the attack should be still less than 2^{80}.

The partial matching details are shown as follows. The column a is for k_a, and b is for k_b. Here we use the same notations as [16] and [17]: *0* implies this bit is fully computable based on information we know and considered as *known*; *1* means computing this bit needs extra key information and is considered as *unknown*. To form a matching, the two resultant bits from both sides should be *known*.

Rd.	a	b	L1	L2
second backward phase				
114	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
113	1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
112	0	0	0 0 0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
111	0	0	0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
110	0	0	0 0 0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
109	0	0	0 0 0 0 0 0 0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
108	0	0	0 0 0 0 0 0 0 0 1 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
107	0	0	0 0 0 0 0 0 0 1 0 0 0 1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
106	0	0	0 0 0 0 0 0 1 0 0 0 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
105	0	0	0 0 0 0 0 1 0 0 0 1 1 0 1 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1
104	1	0	0 0 0 1 0 0 0 1 1 0 1 1 1	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
103	0	1	0 0 1 0 0 0 1 1 0 1 1 1 1	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1
102	0	0	0 1 0 0 0 1 1 0 1 1 1 1 1	0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1
101	0	0	1 0 0 0 1 1 0 1 1 1 1 1 1	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1
100	0	0	0 0 0 1 1 0 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 1
99	0	0	0 0 1 1 0 1 1 1 1 1 1 1 1	0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1
98	1	1	0 1 1 0 1 1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1
97	0	0	1 1 0 1 1 1 1 1 1 1 1 1 1	0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1
96	0	0	1 0 1 1 1 1 1 1 1 1 1 1 1	0 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1
95	0	0	0 1 1 1 1 1 1 1 1 1 1 1 1	1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1
94	1	1	1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1
93	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
92	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1
91	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
90	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
89	1	0	1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
88	0	1	1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
second forward phase				
87	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
matching				
2 bits			1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

For the second MITM attack with partial matching, we cannot simply construct T_2' by using the 2 matching bits and other 46 linear independent values computed based on k_{b_2} as indices, so we propose a simple way using *product set* as in the following attack procedure.

1. Compute $s_{39} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...77}$, and compute $k'_{80...125}$ by treating $k_{78} = k_{79} = 0$. Save $k_{0...77}$ in a table indexed by s_{39} and $k'_{80...125}$. This step is similar to the previous attack. Every entry of the table will have one element on average.
2. Compute the 2 bits of $s'_{87} = D_2(s_{152})$ by using every possible $k_{b_2} = k_{226...303}$, and save all computation results in a table T_2', whose index is $k_{226...303}$.
3. For each guess of s_{63}:
   (a) Compute the 2 bits of $s_{87} = E_2(s_{63})$ by using every possible $k_{f_2} = k_{126...173}$, and store all $k_{126...173}$'s in a table T_2 indexed by different values of the 2 bits, and each entry is a sub-set of $k_{126...173}$'s. After this step, T_2 and T_2' together form a *product set*.
   (b) Compute $s'_{39} = D_1(s_{63})$ for each $k_{b_1} = k_{78...125}$, calculate $k'_{80...125}$ by deducting k_{78} and k_{79} from $k_{80...125}$, and find the matching $k_{0...77}$ in T_1. Next, based on the knowledge of $k_{0...79}$, compute the sub-key pair $k_{126...173}$ and $k_{226...303}$, and check whether the pair is also in the *product set* of T_2 and T_2': First look up $k_{126...173}$ in T_2' to find the corresponding values of the two bits, and then check whether $k_{226...303}$ is in the (set) entry of T_2 indexed by the two bits. If so, perform further brute-force testing on the candidate key.

In this attack we extend the number of rounds to 152. The total time complexity of this attack is

$$2^{78} \cdot \frac{39}{152} + 2^{78} \cdot \frac{65}{152} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{152} + 2^{48} \cdot \frac{24}{152} \right) + 2^{80-2} \approx 2^{79.56}.$$

The memory consumption contains T_1, T_2 and T_2', the total of which is the same as the previous attack, i.e. 2^{79}. Please note that the data complexity of this attack is also the same as before, i.e. 3 known plaintext-ciphertext pairs, because even if the first pair used in the MITM attack phase is only consumed 2-bit information, we can reuse it in the brute-force testing phase to filter out more wrong keys.

3.3 3D-MITM Attacks on KATAN32

For MD-MITM attacks, computations of certain steps may be repeating. For example, as in Fig. 3, E_{f_3} is computed 2^{2n} times, so we can actually cache the computation results for first 2^n times and reuse them later. Let us use this caching technique in attacks on KATAN32, and start from a simple 3D-MITM attack.

The two guessed states are s_{64} and s_{88}. The first MITM attack starts from s_0, ends at s_{64}, and meets at s_{40}. The second MITM attack starts from s_{64}, ends at s_{88}, and meets at s_{80}. The third one is from s_{88} to s_{152}, and meets at s_{112}. The detailed attack procedure is described as follows.

1. Compute $s_{40} = E_1(s_0)$ and $k_{80...127}$ for each possible $k_{f_1} = k_{0...79}$, and store $k_{0...79}$ in a table T_1 indexed by s_{40} and $k_{80...127}$. Every entry of T_1 will have one element on average.
2. Compute $s'_{112} = D_3(s_{152})$ for each $k_{b_3} = k_{224...303}$, and store $k_{224...303}$ in a table T_3' indexed s'_{112}, and each entry of T_3' is a set containing certain $k_{224...303}$'s.
3. For each guessed pair of $g_2 = s_{88}$ and $k_{f_3} = k_{176...223}$, compute $s_{112} = E_3(s_{88})$ and store the computation results in a table T_3 indexed by s_{88} and k_{f_3}. After this step, T_3 and T_3' form a product set.
4. Fore each guess of $g_1 = s_{64}$:
   (a) Compute $s'_{40} = D_1(s_{64})$ for each $k_{b_1} = k_{80...127}$, and find the matching $k_{0...79}$ in T_1 by using the indices $k_{80...127}$ and s'_{40}. Next, based on the matching key $k_{0...79}$ compute $k_{128...175}$, and then store $k_{0...79}$ in a table S indexed by $k_{128...175}$. Each entry of S will have one element on average.

(b) Compute $s_{80} = E_2(g_{64})$ for each $k_{f_2} = k_{128...159}$, and store $k_{128...159}$ in a table T_2 indexed by s_{80}, where each entry has one element on average.
(c) For each guess of $g_2 = s_{88}$:
   i. For each $k_{b_2} = k_{160...175}$, compute $s'_{80} = D_2(s_{88})$, and look up the table T_2 by s'_{80} to find the matching $k_{128...159}$. Next, look up the table S by $k_{128...175}$ to find the matching $k_{0...79}$. Finally use $k_{0...79}$ to compute the corresponding pair of $k_{176...223}$ and $k_{224...303}$, and check whether the pair is also in the product set of T_3 and T'_3. If so, do further brute-force testing on $k_{0...79}$.

The total number of attacked rounds is 152. The time complexity of this 3D-MITM attack is

$$2^{80} \cdot \frac{40}{152} + 2^{80} \cdot \frac{40}{152} + 2^{32+48} \cdot \frac{24}{152} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{152} + 2^{32} \cdot \frac{16}{152} + 2^{32+16} \cdot \frac{8}{152}\right) + 2^{80-32} \approx 2^{79.84}.$$

Since we need to store T_1, T_3, T'_3, S and T_2 in memory, the memory complexity is $2^{80} + 2^{80} + 2^{32+48} + 2^{48} + 2^{32} \approx 2^{81.58}$. We use only one known plaintext-ciphertext pair in the MITM attack phase, so the total data complexity of the 3D-MITM attack is still 3 known plaintexts as before.

Improvements. To reduce the time complexity a little bit and make the memory complexity under 2^{80}, we can lower the numbers of attacked rounds of the phases f_1, f_3 and b_3 by one round. In this case, the memory complexity will decrease to about $2^{81.58-2} = 2^{79.58}$, and the time complexity will also decrease.

The partial matching technique can also be used in 3D-MITM attacks to increase the number of attacked rounds. Adopting the partial matching technique in the phases f_3 and b_3, we can still use the similar positions for the two matching bits as the 2D-MITM attack in the last subsection.

Our final 3D-MITM attack with the above improvement methods can attack 175-round KATAN32. The first MITM attack starts from s_0, ends at s_{63}, and meets at s_{39}. The second one is from s_{63} to s_{87}, and meets at s_{79}. The third one meets at s_{110} and ends at s_{175}. The detailed attack procedure is described as follows.

1. For each possible $k_{f_1} = k_{0...77}$, calculate $s_{39} = E_1(s_0)$, and compute $k'_{80...125}$ by treating $k_{78} = k_{79} = 0$, and store $k_{0...77}$ in a table T_1 indexed by s_{39} and $k'_{80...125}$. Every entry of T_1 will have one element on average.
2. For each $k_{b_3} = k_{272...349}$, compute the 2 bits of $s'_{110} = D_3(s_{175})$ for matching use, and store $k_{272...349}$ in a table T'_3 indexed by values of the two bits. Each entry of T'_3 is a set containing certain $k_{272...349}$'s.
3. For each guessed pair of s_{87} and $k_{f_3} = k_{174...219}$, compute the two bits of $s_{110} = E_3(s_{87})$ and store the computation results in a table T_3 indexed by s_{87} and k_{f_3}. After this step, T_3 and T'_3 form a product set.
4. For each guess of $g_1 = s_{63}$:
   (a) Compute $s'_{39} = D_1(s_{63})$ for each $k_{b_1} = k_{78...125}$, compute $k'_{80...125}$ by deducting k_{78} and k_{79} from $k_{80...125}$, and find the matching $k_{0...77}$ in T_1 by using the indices $k'_{80...125}$ and s'_{39}. Next, compute $k_{126...173}$ based on $k_{0...79}$, and then store $k_{0...79}$ in a table S indexed by $k_{126...173}$. Each entry of S will have one element on average.
   (b) Compute $s_{79} = E_2(s_{63})$ for each $k_{f_2} = k_{126...157}$, and store $k_{126...157}$ in a table T_2 indexed by s_{79}, each entry of which has one element on average.
   (c) For each guess of $g_2 = s_{87}$:
      i. For each $k_{b_2} = k_{158...173}$, compute $s'_{79} = D_2(s_{87})$, look up the table T_2 by s'_{79} to find the matching $k_{126...157}$. Next, look up the table S by $k_{126...173}$ to find the matching $k_{0...79}$. Finally use $k_{0...79}$ to compute the sub-key pair of $k_{174...219}$ and $k_{272...349}$, and check whether the pair is also in the product set of T_3 and T'_3. If so, do further brute-force testing on $k_{0...79}$.

The total time complexity of this attack is

$$2^{78} \cdot \frac{39}{175} + 2^{78} \cdot \frac{65}{175} + 2^{32+46} \cdot \frac{23}{175} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{175} + 2^{32} \cdot \frac{16}{175} + 2^{32+16} \cdot \frac{8}{175} \right) + 2^{80-2} \approx 2^{79.30}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{32+46} + 2^{48} + 2^{32} \approx 2^{79.58}$. The data complexity stays as the same, i.e. 3 known plaintext-ciphertext pairs.

3.4 New Attacks on KATAN48 and KATAN64

We can apply similar 2D-MITM attacks on other versions of KATAN, i.e. KATAN48 and KATAN64, but only one state can be guessed in the middle because the block sizes of KATAN48 and KATAN64 are larger than halves of their key lengths. We can also use the partial matching technique in attacks on KATAN48 and KATAN64 in order to increase numbers of attacked rounds. The detailed analysis procedure is omitted here and we just give the descriptions of the new attacks. The details of partial matching steps are listed in the appendix.

The 2D-MITM attack on KATAN48 can reach 130 rounds. The guessed state is s_{55}. The first MITM attack meets at s_{39} and the second meets at s_{71}. The attack steps are described as follows.

1. Compute $s_{39} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...77}$, and compute $k'_{80...109}$ by treating $k_{78} = k_{79} = 0$. Save $k_{0...77}$ in a table indexed by s_{39} and $k'_{80...125}$.
2. Compute the 2 bits of $s'_{71} = D_2(s_{130})$ by using every possible $k_{b_2} = k_{182...259}$, and save all computation results in a table T'_2, whose index is $k_{182...259}$.
3. For each guess of s_{55}:
   (a) Compute the 2 bits of $s_{71} = E_2(s_{55})$ by using every possible $k_{f_2} = k_{110...141}$, and store all $k_{110...141}$'s in a table T_2 indexed by values of the 2 bits, and each entry is a sub-set containing certain $k_{110...141}$'s. After this step, T_2 and T'_2 together form a product set.
   (b) Compute $s'_{39} = D_1(s_{55})$ for each $k_{b_1} = k_{78...109}$, calculate $k'_{80...109}$ by deducting k_{78} and k_{79} from $k_{80...109}$, and find the matching $k_{0...77}$ in T_1. Next, based on the knowledge of $k_{0...79}$, compute the sub-key pair of $k_{110...141}$ and $k_{182...259}$, and check whether the pair is also in the product set of T_2 and T'_2. If so, do further brute-force testing on the candidate key.

The time complexity is

$$2^{78} \cdot \frac{39}{130} + 2^{78} \cdot \frac{59}{130} + 2^{48} \cdot \left(2^{32} \cdot \frac{16}{130} + 2^{32} \cdot \frac{16}{130} \right) + 2^{80-2} \approx 2^{79.45}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{32} \approx 2^{79}$. The data complexity is $\lceil 80/48 \rceil = 2$ known plaintext-ciphertext pairs.

The new attack on KATAN64 is similar as above, except we found performing partial matching technique in the first MITM attack will allow us attack more rounds. The final number of attacked rounds on KATAN64 is 112. The guessed point is s_{65}, the first MITM attack meets at s_{46}, and the second one meets at s_{73}. The attack steps are as follows, and the detailed partial matching procedure is listed in the appendix.

1. Compute the 2 bits of $s_{46} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...77}$, and save all computation results in a table T_1 indexed by $k_{0...77}$.
2. Compute $s'_{73} = D_2(s_{112})$ by using every possible $k_{b_2} = k_{146...222}$, and compute $k'_{130...143}$ by treating $k_{144} = k_{145} = 0$. Save $k_{146...222}$ in a table T'_2 indexed by s'_{72} and $k'_{130...143}$.
3. For each guess of s_{65}:
   (a) Compute the 2 bits of $s'_{46} = D_1(s_{65})$ by using every possible $k_{b_2} = k_{114...129}$, and store all $k_{114...129}$'s in a table T'_1 indexed by values of the 2 bits, and each entry is a sub-set containing certain $k_{114...129}$'s. After this step, T_1 and T'_1 together form a product set.

(b) Compute $s_{73} = E_2(s_{65})$ for each $k_{f_2} = k_{130...145}$, calculate $k'_{130...143}$ by deducting k_{144} and k_{145} from $k_{130...143}$, and find the matching $k_{146...222}$ in T'_2. Next, based on the knowledge of $k_{144...223}$, compute the sub-key pair of $k_{0...77}$ and $k_{114...129}$, and check whether the pair is also in the product set of T_1 and T'_1. If so, do further brute-force testing on the candidate key.

The time complexity is

$$2^{78} \cdot \frac{46}{112} + 2^{78} \cdot \frac{39}{112} + 2^{48} \cdot \left(2^{32} \cdot \frac{19}{112} + 2^{32} \cdot \frac{8}{112}\right) + 2^{80-2} \approx 2^{79.45}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{16} \approx 2^{79}$. The data complexity is $\lceil 80/64 \rceil = 2$ known plaintext-ciphertext pairs.

4 Optimization Methods

The new attacks proposed in the previous section focus on maximizing the numbers of rounds that we can attack. There are many techniques that can help us reducing the attacks' complexities.

One way to reduce the time complexities is that when computing intermediate states for partial matching we do not actually need to complete the calculations of partial encryptions or decryptions. Consider the detailed steps of the partial matching used in the 2D-MITM attack on KATAN32 (see Sec 3.2). One of the two bits used for matching in s'_{87} has actually been computed after the decryption of the 106th round, and the other bit is computed in the 104th round. So we do not need to continue the partial decryptions after the 104th round. Moreover, the computations of these two bits depend on only parts of previous states, and thus we may also be able to save some time on the computations before the 104th round. But this technique will not push our attacks to more rounds, and might make attack steps very complicated to explain. Based upon these considerations, in addition to making our complexity estimations generous, this optimization method is not used in our attacks.

Another way to improve the attacks is to segment the ciphers' round functions into smaller steps. For example, the round functions of KATAN48 and KATAN64 update the internal states by two and three times respectively, by using same sub-keys, so we may divide them to two or three sub-functions. And we can even separate operations of updating L_1 and L_2 to different sub-steps, which is applicable to any KATAN variant. By analyzing iterations of smaller steps or functions, we may further refine time and memory complexities, or extend attacks to more rounds.

The paper [17] proposes an improved partial matching technique called *indirect partial matching*, in order to obtain more usable intermediate bits for matching. Originally, when computing a partial matching state, if the value of one bit s_i depends on the key bit k_j only known to its opposite phase, then s_i will be considered as *unknown*. Nonetheless, after adding this key bit k_j into computations, k_j may still remain linear in intermediate states for many rounds. Thus, if we compute $s_i \oplus k_j$ instead of s_i, this bit can still be used for matching. This technique may help us extend our attacks to more rounds, which can be a future work.

5 Concluding Remarks

In this paper, we investigate a new cryptanalysis method called multidimensional meet-in-the-middle attack. A theoretical analysis is given and actual attack examples are presented on the block cipher family KATAN. The new attacks on KATAN32/48/64 can reach much more numbers of rounds than existing attacks. Multidimensional meet-in-the-middle attacks are very applicable to lightweight ciphers with simple key scheduling algorithms and block sizes smaller than master key sizes.

Consider KATAN's sibling block cipher family KTANTAN32/48/64 [1], who has the same round functions as KATAN but a different key scheduling algorithm. For its key schedule algorithm, we cannot manually find a MD-MITM attack on it. However, we may be able to refine our matching techniques,

and program to search potential attack patterns, in which way we may able to apply more efficient attacks on KTANTAN. Similarly, we may apply MD-MITM attacks to other lightweight ciphers.

For MD-MITM attacks with dimensions larger than two, there are portions of the attacks that can be pre-computed. For example, the two ends of the middle portion for a 3D-MITM attack are both guessed values. Thus we can build a look-up table for the middle computations off-line without any knowledge about plaintexts and ciphertexts, which is illustrated in Fig. 5. Especially, we can use any means to construct the loop-up table, not only limited to MITM methods.

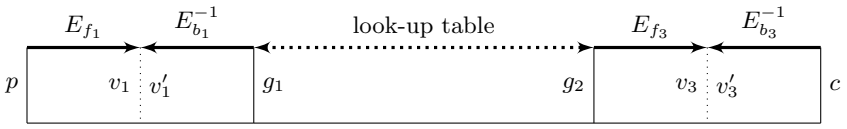

Fig. 5. Multidimensional MITM attacks with look-up tables.

References

1. Cannière, C.D., Dunkelman, O., Knezevic, M.: Katan and ktantan - a family of small and efficient hardware-oriented block ciphers. In Clavier, C., Gaj, K., eds.: CHES. Volume 5747 of Lecture Notes in Computer Science., Springer (2009) 272–288
2. Engels, D., Saarinen, M.J.O., Schweitzer, P., Smith, E.M.: The hummingbird-2 lightweight authenticated encryption algorithm. Cryptology ePrint Archive, Report 2011/126 (2011) `http://eprint.iacr.org/`.
3. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: Printcipher: A block cipher for ic-printing. In Mangard, S., Standaert, F.X., eds.: CHES. Volume 6225 of Lecture Notes in Computer Science., Springer (2010) 16–32
4. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The led block cipher. In Preneel, B., Takagi, T., eds.: CHES. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 326–341
5. Diffie, W., Hellman, M.: Exhaustive cryptanalysis of the nbs data encryption standard. Computer **10**(6) (1977) 74–84
6. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full aes. Cryptology ePrint Archive, Report 2011/449 (2011) `http://eprint.iacr.org/`.
7. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-bicliques: Cryptanalysis of full idea. In Pointcheval, D., Johansson, T., eds.: Advances in Cryptology EUROCRYPT 2012. Volume 7237 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) 392–410
8. Sasaki, Y., Aoki, K.: Finding preimages in full md5 faster than exhaustive search. In Joux, A., ed.: EUROCRYPT. Volume 5479 of Lecture Notes in Computer Science., Springer (2009) 134–152
9. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced sha-0 and sha-1. In Halevi, S., ed.: CRYPTO. Volume 5677 of Lecture Notes in Computer Science., Springer (2009) 70–89
10. Courtois, N.T.: Algebraic complexity reduction and cryptanalysis of gost (2011) `http://www.nicolascourtois.com/papers/gostac11.pdf`.
11. Dinur, I., Dunkelman, O., Shamir, A.: Improved attacks on full gost. Cryptology ePrint Archive, Report 2011/558 (2011) `http://eprint.iacr.org/`.
12. Isobe, T.: A single-key attack on the full gost block cipher. In Joux, A., ed.: Fast Software Encryption. Volume 6733 of Lecture Notes in Computer Science. Springer (2011) 290–305
13. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Advances in Cryptology - Crypto 2012. Volume 7417 of Lecture Notes in Computer Science., Springer (2012) 719–740
14. Albrecht, M., Leander, G.: An all-in-one approach to differential cryptanalysis for small block ciphers. Cryptology ePrint Archive, Report 2012/401 (2012) `http://eprint.iacr.org/`.
15. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. To appear in the Proceedings of Selected Areas in Cryptography (SAC 2012), Windsor, August 16-17, 2012.

16. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher ktantan. Cryptology ePrint Archive, Report 2010/532 (2010) `http://eprint.iacr.org/`.
17. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved meet-in-the-middle cryptanalysis of ktantan. Cryptology ePrint Archive, Report 2011/201 (2011) `http://eprint.iacr.org/`.

Appendix A Parameters for the KATAN Family of Block Ciphers

The parameters for the nonlinear functions (3) are given in Table 2. The irregular update sequence (IR) is listed in Table 3.

Table 2. Parameters for KATAN.

	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

Table 3. Irregular Update Sequence (IR) for KATAN.

#round	IR
1 - 20	1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1
21 - 40	1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0
41 - 60	0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0
61 - 80	0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1
81 - 100	1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1
101 - 120	0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1
121 - 140	1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0
141 - 160	1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1
161 - 180	1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0
181 - 200	0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0
201 - 220	1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0
221 - 240	0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1
241 - 254	1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0

Appendix B Detailed Steps of Partial-matching for KATAN48/64

The detailed computation steps of partial matching technique used in the second MITM phase of the attack on KATAN48 are listed as follows.

```
Rd.  a b  L1                                     L2
second backward phase
92   0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
91   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
90   0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
89   0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
88   0 0  0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1
87   0 0  0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1
86   0 0  0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0
85   0 0  0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1
84   0 0  0 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1
83   0 0  0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1
82   1 0  1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1
81   0 1  1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1
```

```
80   0 0  0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
79   0 0  1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
78   0 0  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
77   0 0  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
76   1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
75   0 0  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
74   0 0  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
73   0 0  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
72   1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
second forward phase
71   0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
matching
2 bits    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The steps of partial matching for the first MITM phase of the attack on KATAN64 are listed as follows.

```
Rd.  a b  L1                                          L2
first forward phase
39   0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
40   1 1  1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
41   0 0  0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
42   0 0  0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
43   0 0  1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
44   0 0  0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0  1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
45   0 0  1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0  1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
46   0 1  1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1  1 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
first backward phase
58   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
57   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
56   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
55   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
54   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
53   1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
52   1 1  0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
51   1 1  0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
50   1 1  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
49   1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
48   1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
47   1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
matching
2 bits    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```