Noname manuscript No.
(will be inserted by the editor)

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

Bo Zhu · Guang Gong

the date of receipt and acceptance should be inserted later

Abstract This paper investigates a new framework to analyze symmetric ciphers by guessing intermediate states and dividing algorithms into consecutive sub-ciphers. It is suitable for lightweight ciphers with simple key schedules and block sizes smaller than key lengths. New attacks on the block cipher family KATAN are proposed by adopting this framework. Our new attacks can recover the master keys of 175-round KATAN32, 130-round KATAN48 and 112-round KATAN64 faster than exhaustive search, and thus reach many more rounds than previous attacks. We also provide new attacks on 115-round KATAN32 and 100-round KATAN48 in order to demonstrate this new kind of attacks can be more time-efficient and memory-efficient than existing attacks.

Keywords Multidimensional · Meet-in-the-middle · Cryptanalysis · KATAN

1 Introduction

Lightweight devices such as NFC/RFID chips and wireless sensor networks have become popular nowadays, because such tiny devices bring more convenience to people's 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, traditional designs of cryptographic algorithms and protocols, such as TLS/SSL, can hardly be used in such environments 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 [5], a lightweight stream cipher WG-7 [17], and the authenticated encryption algorithm Hummingbird-2 [10] are devised specifically for constrained environments. The block cipher PRINTcipher [16] is designed to be compact enough for integrated circuit printing. A 64-bit version block cipher, LED [11], is proposed based on the structure of AES, which has similar security evaluation but smaller implementation footprints. Security evaluation of these lightweight algorithms should be a very important work for researchers.

Bo Zhu · Guang Gong
University of Waterloo, Waterloo, Ontario, Canada
e-mail: {bo.zhu,ggong}@uwaterloo.ca

Meet-in-the-middle (MITM, hereafter) attack was first introduced by Diffie and Hellman in [7] for cryptanalysis of DES, and it 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 smaller 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 combine 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.

Our Contributions. Inspired by the recent development of MITM techniques for cryptanalysis, such as biclique attacks [3, 15] that create the first single-key attacks on full AES and IDEA, and splice-and-cut attacks for computing pre-images of MD5 [18], SHA-0 and SHA-1 [2], here we investigate a new method in depth: Ciphers are first divided into consecutive sub-ciphers by guessing certain intermediate states, then MITM attacks are applied to these sub-ciphers separately, and finally results are brought together to eliminate wrong keys. We apply this multidimensional approach to the block cipher family KATAN, and obtain the best cryptanalysis results so far. Our new attacks can recover the master keys of 175-round KATAN32, 130-round KATAN48, and 112-round KATAN64 faster than exhaustive search, which reach many more round than existing attacks. New attacks on 115-round KATAN32 and 100-round KATAN48 are also proposed in order to show that this new kind of attacks can be more efficient than existing results.

Related Work. The papers [6,9] have the basic idea of guessing one internal state in MITM attacks, but their attacks only succeed in improving memory and data complexities, but not time complexity, of the previous work in [12]. We also note that there is an independent work [8] with similar ideas, but the authors focus on optimizing time-memory trade-offs for composite problems, and their analysis is only applied to the cases where all sub-ciphers have independent keys.

Our new cryptanalysis results on KATAN are summarized in Table 1, where the notation KP stands for *known plaintexts*.

Table 1 Comparisons of previous and new cryptanalysis results on reduced-round KATAN ciphers.

Cipher	Rounds	Time Compl.	Memory Compl.	Data Compl.	Reference
KATAN32	115	2^{79}	Not Given	2^{32} KP	[1]
	115	$2^{77.75}$	$2^{72.32}$	3 KP	Sec. 6.1
	119	$2^{79.10}$	$2^{79.10}$	144 KP	[13]
	175	$2^{79.30}$	$2^{79.58}$	3 KP	Sec. 3.2.1
KATAN48	100	2^{78}	2^{78}	128 KP	[14]
	100	$2^{77.37}$	$2^{73.32}$	2 KP	Sec. 6.2
	105	$2^{79.10}$	$2^{79.10}$	144 KP	[13]
	130	$2^{79.45}$	$2^{79.00}$	2 KP	Sec. 4
KATAN64	94	$2^{77.68}$	$2^{77.68}$	116 KP	[14]
	99	$2^{79.10}$	$2^{79.10}$	142 KP	[13]
	112	$2^{79.45}$	$2^{79.00}$	2 KP	Sec. 4

2 Multidimensional Meet-in-the-Middle Attack

This section will first briefly introduce the original MITM attack, and then discuss how to extend it in a multidimensional approach.

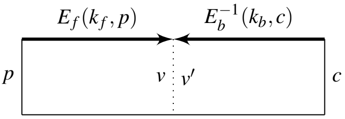

Fig. 1 An illustration of meet-in-the-middle attacks.

2.1 Meet-in-the-Middle Attack

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

$$c = 2\mathrm{DES}_{(k_1,k_2)}(p) = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(p)) .$$

The total number of key bits is $2 \cdot 56 = 112$, so the time complexity of the exhaustive key search for 2DES is 2^{112}. To launch a MITM attack, firstly, we compute $v = \mathrm{DES}_{k_1}(p)$ 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 2^{56}. Secondly, from the ciphertext side, we compute the decryption $v' = \mathrm{DES}_{k_2}^{-1}(c)$ 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 evaluate DES for $2 \cdot 2^{56} = 2^{57}$ times, which is much less than 2^{112}. This is the reason why we should use Triple-DES, rather than Double-DES, to obtain a reasonably larger security margin than DES.

More formally, assume 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 subkeys used in E_f and E_b, as shown in Fig. 1. Here f and b are the abbreviations for *forward* and *backward*. The steps of MITM attacks can be written as follows.

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

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 ciphers' intermediate states are smaller than block sizes, e.g., $|v| \leq n$, in the following content. The time complexity for Step 1.1 is $2^{|k_f|}$, and for Step 1.2 it is $2^{|k_b|}$. During the MITM phase, wrong keys have the probability of $1/2^{|v|}$ to obtain a false positive. Thus if k_f and k_b do not have common key bits, the number of wrong keys passing the MITM phase will be $2^{|k_f|+|k_b|}/2^{|v|} = 2^{|k_f|+|k_b|-|v|}$. If k_f and k_b have common key bits, we let k_c denote all the key bits contained in both k_f and k_b, so the number of remaining keys will be $2^{|k_f|+|k_b|-|k_c|-|v|}$. We further assume k is the master key that consists of all the key bits of k_f and k_b, and $|v|$ is equal to the block size n. Then we have

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

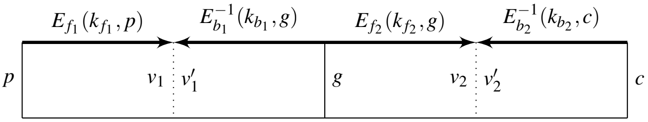

Fig. 2 Meet-in-the-middle attacks with one guess.

This can be easily understood from the information theory's point of view: Since we have the information of an n-bit plaintext-ciphertext pair (p, c), we can only reduce the key space to $1/2^n$ of the original. After this MITM phase, we can simply deploy brute-force testing to remove 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 plaintext-ciphertext pair 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 of the MITM attack 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 [4].

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 maintaining the set S. There are many kinds of data structures to construct S, such as hash tables. Actually, the construction 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 cipher encryption. We suggest using 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, the memory complexity of the MITM attack should be $2^{|k_f|}$ or $2^{|k_b|}$ at least.

2.2 Multidimensional Meet-in-the-Middle Attack

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 kind of designs is perfectly valid, it is possible to guess certain short intermediate states and divide the ciphers into small sub-ciphers for easier analysis.

Let us first give a simple and rather inefficient attack framework for ease of understanding. In following sections, refined methods will be given in real attacks on the KATAN block cipher family.

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

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 $v'_2 = E^{-1}_{b_2}(k_{b_2}, c)$ for each possible k_{b_2}, and put all k_{b_2}'s into a table T'_2 (similar as T_1) indexed by v'_2.
3. For each possible guess of g:
   (a) Compute $v'_1 = E^{-1}_{b_1}(k_{b_1}, g)$ 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.

Since we do not need to recompute E_{f_1} and $E^{-1}_{b_2}$ for different g's, the time complexity of this attack without the brute-force testing phase is

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

For each guessed value of g, the MITM step from p to g will reduce the size of the key space to $2^{|k|-|v_1|}$ and the second MITM step for the interval between g and c 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 around $2^{|k|-n}$.

Please note a subtle part in the above analysis: Although the size of the master key space is reduced to $2^{|k|-|v_1|}$ after the MITM attack step from p to g, the number of sub-keys to be matched with the results from the other MITM step between g and c is only $2^{|k_{f_1}|+|k_{b_1}|-|v_1|}$, which may be much less than $2^{|k|-|v_1|}$.

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 may need to store T_1, T'_1, T_2 and T'_2 in memory. The data complexity of the attack is $\lceil |k|/n \rceil$.

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 treating each sub-key bit as an independent new variable. This technique has been used in other cryptanalysis methods, such as [14]. But we may get more efficient results or attack more rounds by carefully investigating ciphers' detailed designs. For example, we may perform linear transformations before matching sub-keys, or study round functions to perform partial encryptions/decryptions. We will show real attack examples using these techniques in following sections.

Certainly, we can guess more intermediate states and then segment ciphers into smaller pieces. 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 $v'_{i+1} = E^{-1}_{b_{i+1}}(k_{b_{i+1}}, c)$.

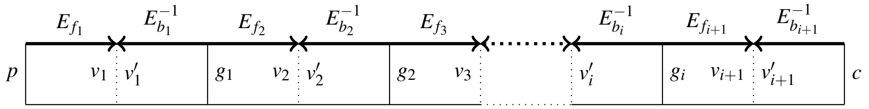

Fig. 3 General process of multidimensional meet-in-the-middle attacks with multiple guesses.

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 $v_2 = E_{f_2}(k_{f_2}, g_1)$.
   (c) For each guess of g_2:
      i. Construct a table T'_2 by computing $v'_2 = E_{b_2}^{-1}(k_{b_2}, g_2)$, 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} in order 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 combination as the correct key.

If we assume $|g_1| = |g_2| = \cdots = n$, the time complexity of the MITM phase with multiple guesses is

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

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 order to obtain more desirable results.

The memory complexities of MD-MITM attacks are upper-bounded by the total memory consumption of $T_1, T'_1, T_2, \cdots, T'_{i+1}$. In order to obtain the minimum value for the time complexity equation (1), 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 $2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|}$.

We only use one known plaintext-ciphertext pair before the brute-force testing phase, so it is easy to see that MD-MITM attacks have the same data complexities as 1D-MITM attacks, i.e. $\lceil |k|/n \rceil$ known plaintext-ciphertext pairs.

What we want to emphasize here is that the theoretical analysis in this section only presents a general framework to perform the divide-and-conquer method on ciphers. Whether it can really work and improve attacks' efficiencies on a specific cipher depends on the detailed specifications of the cipher. In the next sections, we will demonstrate several MD-MITM attacks on the block cipher family KATAN32/48/64, which can attack more rounds and reduce complexities of previous attacks. These new attacks highly rely on certain design details of KATAN.

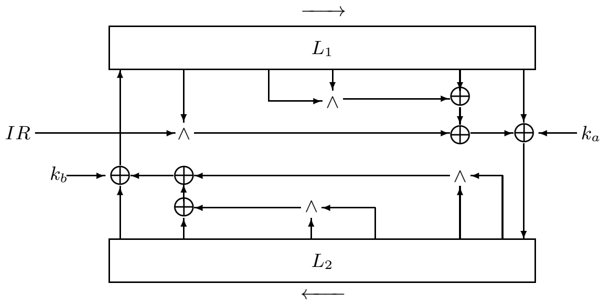

Fig. 4 The structure of KATAN [5].

3 Application to KATAN32

KATAN [5] is a block cipher family designed for constrained devices and embedded systems. It consists of three versions with different block sizes, 32, 48 and 64 bits, which are named KATAN32, KATAN48 and KATAN64 respectively. Despite of the 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 (2) are operated on L_1 and L_2 respectively.

$$\begin{aligned} f_a[L_1] &= L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a \\ f_b[L_2] &= L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b \end{aligned} \tag{2}$$

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 and IR are given in the appendix.

k_a and k_b are two sub-key bits produced from a 80-bit master key K by a linear feedback shift register (LFSR). The master key K is loaded as the initial state of the LFSR, and each output bit of the LFSR is used as a sub-key bit sequentially. Assuming $\{k_i\}$ is the output sequence of the LFSR, if $i < 80$, k_i is equal to the i-th bit of the master key; if $i \geq 80$,

$$k_i = k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13}. \tag{3}$$

For the r-th round of KATAN, $k_a = k_{2r-2}$ 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 by one bit, 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 operations, the states of L_1 and L_2 are concatenated and output as the ciphertext c.

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

3.1 2D-MITM Attacks on KATAN32

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

1. Compute $s_{40} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...79}$, and compute $k_{80...127}$ from k_{f_1} by using linear functions derived from the LFSR (see the equation (3)). Put each k_{f_1} 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 $s'_{88} = D_2(s_{128})$ by using every possible $k_{b_2} = k_{176...255}$, and compute $k_{128...175}$ from k_{b_2} by using linear functions derived from the LFSR. Store each k_{b_2} 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 $s'_{40} = D_1(g)$ for each $k_{b_1} = k_{80...127}$, and then find the matching key k_{f_1} in T_1. On average there is only one such key, and we put it in a set S.
   (b) Compute $s_{88} = E_2(g)$ for each $k_{f_2} = k_{128...175}$, and find the matching key k_{b_2} in T'_2. Then compute $k_{0...79}$ by using linear equations derived from the LFSR, and check whether it is 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 time complexities of existing attacks, we adopt the formula proposed in [19] 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 phases of the 2D-MITM attack, and R to denote the total number of attacked rounds. The time complexity without the brute-force testing phase is computed as follows.

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

The above equation can be seen as estimating the equivalent number of full R-round cipher evaluations, as the time to complete the 2D-MITM attack.

Here we simply ignore the time complexities of linear transformations when matching sub-keys, because these LFSR computations only involve several linear operations, which are more cost-efficient compared to iterations of nonlinear round functions, and the exhaustive key search also needs to compute the sub-keys and will consume equivalent time.

In the above 2D-MITM attack on KATAN32, since $|k_{f_1}| = |k_{b_2}| = 80$ and $|k_{b_1}| = |k_{f_2}| = 48$, 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. $\lceil 80/32 \rceil = 3$ plaintext-ciphertext pairs.

3.1.1 Reducing Time Complexities

To make the time complexity of the above attack better than exhaustive search, i.e. 2^{80}, we can reduce the numbers of attacked rounds in the first forward and second backward phases by one. But in this case, when constructing T_1 (or similarly T_2'), we cannot simply use the intermediate state $g = s_{38}$ and $k_{b_1} = k_{78...125}$ as indices like in the previous 2D-MITM attack, because the new $k_{f_1} = k_{0...77}$ does not have the full 80-bit information of the master key $K = k_{0...79}$, and certain bits of k_{b_1} still depend on the values of k_{78} and k_{79}.

However, by assuming k_{78} and k_{79} are zero, we can compute a temporal key $\kappa_{80...125}$ for the purpose of matching, which is equivalent to removing these two bits from the linear equations of key scheduling. Under such circumstance, the first MITM phase between p and g can be performed without knowing k_{78} and k_{79}, and later extra information of k_{78} and k_{79} can be appended to form the 80-bit master key. The detailed attack procedure is described as follows.

1. Compute $s_{39} = E_1(s_0)$ by iterating every possible $k_{f_1} = k_{0...77}$, and compute $\kappa_{80...125}$ from $k_{0...77}$ by assuming $k_{78} = k_{79} = 0$. Put each k_{f_1} in a table T_1 indexed by s_{39} and $\kappa_{80...125}$. Each entry of the table should have one element on average.
2. Compute $s'_{87} = D_2(s_{126})$ using every possible $k_{b_2} = k_{174...251}$, and compute $\kappa_{126...171}$ from $k_{174...251}$ by treating $k_{172} = k_{173} = 0$. Store each k_{b_2} in a table T_2' indexed by s'_{87} and $\kappa_{126...171}$. Similarly, each entry of the table has one element on average.
3. For each guess of $g = s_{63}$:
   (a) Compute $s'_{39} = D_1(g)$ for each $k_{b_1} = k_{78...125}$, and compute $\kappa_{80...125}$ by subtracting k_{78} and k_{79} from the bits of $k_{80...125}$. Then use s'_{39} and $\kappa_{80...125}$ to find the matching k_{f_1} in T_1. On average there is only one matching $k_{0...77}$, 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 $s_{87} = E_2(g)$ for each $k_{f_2} = k_{126...173}$, and compute $\kappa_{126...171}$ by subtracting k_{172} and k_{173} from $k_{126...172}$. Then use s_{87} and $\kappa_{126...171}$ to find the matching k_{b_2} in T_2'. Compute $k_{0...79}$ from $k_{172...251}$ by using linear functions derived from the LFSR, and check whether it is 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 more number of rounds than any previous attack on KATAN32, but we still have room for improvements.

3.1.2 Increasing the Number of Attacked Rounds

We can see there is a large time complexity gap between the MD-MITM and brute-force testing phases in the above 2D-MITM attacks on KATAN32. The time complexities of brute-force testing are always $2^{80-32} = 2^{48}$, 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 we could 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 [4,19].

By adopting the partial matching technique, we can extend our attack on 126 rounds of KATAN32 to 152 rounds. For simplicity, we only use partial matching in the second MITM part. After searching by programs, we find the best position for the second backward phase is from s_{152} to s_{87}. 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 [4] and [19]: *0* implies this bit is fully computable based on information we know and thus 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
113   1 1  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 1
112   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 1 0
111   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 1 0 0
110   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 1 0 0 0
109   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 1 0 0 0 0
108   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 1 0 0 0 0 0
107   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 1 0 0 0 0 0 1
106   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 1 0 0 0 0 0 1 0
105   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 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 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 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 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 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 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
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, along with a 46-bit temporal key computed from k_{b_2}, as indices, so we propose using a *product set* that is constructed by two related tables: First find an index value from the first table, and then locate the target value from the second table by using the index value. The attack procedure is described as follows.

1. By using every possible $k_{f_1} = k_{0...77}$, compute $s_{39} = E_1(s_0)$, and calculate $\kappa_{80...125}$ by treating $k_{78} = k_{79} = 0$. Save k_{f_1} in a table indexed by s_{39} and $\kappa_{80...125}$. 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_{b_2}.
3. For each guess of $g = s_{63}$:
   (a) Compute the 2 bits of $s_{87} = E_2(g)$ for every possible $k_{f_2} = k_{126...173}$, and store all k_{f_2}'s in a table T_2 indexed by different values of the 2 bits. Each entry of T_2 is a

sub-set of k_{f_2}'s. After this step, T_2 and T_2' together form a *product set*, and we will show how to look up its elements in next steps.

(b) For each $k_{b_1} = k_{78...125}$:
   i. Compute $s_{39}' = D_1(g)$. Calculate $\kappa_{80...125}$ from $k_{80...125}$ by subtracting k_{78} and k_{79}. Use s_{39}' and $\kappa_{80...125}$ to find the matching k_{f_1} in T_1. Now we will have the (guessed) full 80-bit information of $k_{0...79}$.
   ii. Based on the knowledge of $k_{0...79}$, compute the sub-key pair k_{f_2} and k_{b_2}, and check whether the pair is also in the product set of T_2 and T_2': First look up k_{b_2} in T_2' to find the corresponding values of the two bits, and then check whether k_{f_2} is in the entry (set) of T_2 indexed by the two bits. If so, perform further brute-force testing on the candidate key.

To further explain the computation of temporal sub-keys, we take $\kappa_{80...125}$ in Step 3(b)(i) for example: Since each bit of $k_{80...125}$ can be expressed by a linear function in terms of the bits of $k_{0...79}$, we can subtract the values of k_{78} and k_{79} from these linear expressions to get the temporal key $\kappa_{80...125}$, only for the purpose of matching.

The total time complexity of this 2D-MITM 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', and the total 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 phase is only consumed by 2-bit information, we can reuse it in the brute-force testing phase to filter out more wrong keys.

3.2 3D-MITM Attacks on KATAN32

For MD-MITM attacks, computations of certain steps may have been repeated. For example, as in Fig. 3, E_{f_3} is computed for 2^{2n} times, so we may cache computation results of first 2^n times and reuse them later.

As previous subsections, we first start from a simple 3D-MITM attack. The two guessed states are $g_1 = s_{64}$ and $g_2 = s_{88}$. The first MITM part starts from s_0, ends at g_1, and meets at s_{40}. The second MITM part starts from g_1, ends at g_2, and meets at s_{80}. The third one is from g_2 to s_{152}, and meets at s_{112}. The key point of this attack is to keep the numbers of the key bits of k_{f_2} and k_{b_2} small enough, such that we can expect only one sub-key in each intermediate matching step, in order to keep the attack simple. The detailed attack procedure is described as follows.

1. Compute $s_{40} = E_1(s_0)$ and $k_{b_1} = k_{80...127}$ for each possible $k_{f_1} = k_{0...79}$, and store k_{f_1} in a table T_1 indexed by s_{40} and k_{b_1}. 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_{b_3} in a table T_3' indexed s_{112}'. Each entry of T_3' is a set containing certain k_{b_3}'s.
3. For each guessed pair of $g_2 = s_{88}$ and $k_{f_3} = k_{176...223}$, compute $s_{112} = E_3(g_2)$ and store the computation results in a table T_3 indexed by g_2 and k_{f_3}. After this step, T_3 and T_3' form a product set.
4. For each guess of $g_1 = s_{64}$:

(a) Compute $s'_{40} = D_1(g_1)$ for each $k_{b_1} = k_{80\ldots127}$, and find the matching $k_{f_1} = k_{0\ldots79}$ in T_1 by using the indices k_{b_1} and s'_{40}. Next, based on the 80-bit master key $k_{0\ldots79}$, we compute $k_{128\ldots175}$, and then store $k_{0\ldots79}$ in a table S indexed by $k_{128\ldots175}$. Each entry of S will have one element on average.

(b) Compute $s_{80} = E_2(g_1)$ for each 32-bit $k_{f_2} = k_{128\ldots159}$, and store k_{f_2} in a table T_2 indexed by s_{80}, where each entry has one element on average.

(c) For each guess of $g_2 = s_{88}$:

- For each 16-bit $k_{b_2} = k_{160\ldots175}$:
    1. Compute $s'_{80} = D_2(s_{88})$, and look up the table T_2 by s'_{80} to find the matching $k_{f_2} = k_{128\ldots159}$. Thus we get a consecutive 48-bit sub-key $k_{128\ldots175}$.
    2. Look up the table S by $k_{128\ldots175}$ to find the matching $k_{0\ldots79}$.
    3. Finally use $k_{0\ldots79}$ to compute the corresponding pair of k_{f_3} and k_{b_3}, 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 $k_{0\ldots79}$.

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

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

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 3 known plaintext-ciphertext pairs as before.

3.2.1 Improvements

To make the memory complexity become under 2^{80}, we can lower the numbers of attacked rounds in the phases f_1, f_3 and b_3 by one. In this case, its memory complexity will decrease to about $2^{81.58-2} = 2^{79.58}$, and the time complexity will decrease as well.

The partial matching technique can also be used in 3D-MITM attacks to increase the number of attacked rounds. Adopting partial matching 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 on KATAN32 can reach 175 rounds at most. The first MITM part starts from s_0, meets at s_{39}, and ends at $g_1 = s_{63}$. The second one is from g_1 to $g_2 = s_{87}$, and meets at s_{79}. The third one meets at s_{110} and ends at s_{175}. The overall attack is similar to the previous 3M-MITM one, except partial matching is employed in the third MITM part. The detailed attack procedure is described as follows.

1. For each possible $k_{f_1} = k_{0\ldots77}$, calculate $s_{39} = E_1(s_0)$, and compute $\kappa_{80\ldots125}$ by treating $k_{78} = k_{79} = 0$. Store k_{f_1} in a table T_1 indexed by s_{39} and $\kappa_{80\ldots125}$. Every entry of T_1 will have one element on average.
2. For each $k_{b_3} = k_{272\ldots349}$, compute the 2 bits of $s'_{110} = D_3(s_{175})$ for matching use, and store k_{b_3} in a table T'_3 indexed by values of the two bits. Each entry of T'_3 is a set containing certain k_{b_3}'s.
3. For each guessed pair of $g_2 = s_{87}$ and $k_{f_3} = k_{174\ldots219}$, compute the two bits of $s_{110} = E_3(s_{87})$ and store the computation results in a table T_3 indexed by g_2 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 $s'_{39} = D_1(g_1)$ for each $k_{b_1} = k_{78...125}$. Compute $\kappa_{80...125}$ from $k_{80...125}$ by subtracting k_{78} and k_{79}. Find the matching k_{f_1} in T_1 by using the indices $\kappa_{80...125}$ 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 $s_{79} = E_2(g_1)$ for each $k_{f_2} = k_{126...157}$, and store k_{f_2} in a table T_2 indexed by s_{79}, each entry of which has one element on average.

(c) For each guess of $g_2 = s_{87}$:
   - For each 16-bit $k_{b_2} = k_{158...173}$:
     i. Compute $s'_{79} = D_2(g_2)$, and look up the table T_2 by s'_{79} to find the matching $k_{f_2} = k_{126...157}$. Now we will have the complete 48-bit information about $k_{126...173}$.
     ii. Look up the table S by $k_{126...173}$ to find the matching $k_{0...79}$.
     iii. Finally use $k_{0...79}$ to compute the sub-key pair of k_{f_3} and k_{b_3}, 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

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

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.

4 Applications to KATAN48 and KATAN64

We can apply similar MD-MITM attacks to other versions of KATAN, i.e. KATAN48 and KATAN64, but only one intermediate state g can be guessed because the block sizes of KATAN48 and KATAN64 are larger than halves of their key lengths. We can also use the partial matching technique here in order to increase numbers of attacked rounds. We omit the detailed analysis procedure and just give 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 at most. The guessed state is $g = s_{55}$. The first MITM part meets at s_{39}, and the second meets at s_{71}. The partial matching technique is used in the second MITM part. 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 $\kappa_{80...109}$ by treating $k_{78} = k_{79} = 0$. Save $k_{0...77}$ in a table indexed by s_{39} and $\kappa_{80...125}$.
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_{b_2}.
3. For each guess of $g = s_{55}$:

   (a) Compute the 2 bits of $s_{71} = E_2(g)$ by using every possible $k_{f_2} = k_{110...141}$. Store all k_{f_2}'s in a table T_2 indexed by values of the 2 bits, and each entry is a sub-set containing certain k_{f_2}'s. After this step, T_2 and T'_2 together form a product set.

   (b) Compute $s'_{39} = D_1(g)$ for each $k_{b_1} = k_{78...109}$, calculate $\kappa_{80...109}$ by subtracting 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_{f_2} and k_{b_2}, and check whether the pair is 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 after searching by programs we find it will allow us to attack more rounds if the partial matching technique is performed in the first MITM part. The final number of attacked rounds on KATAN64 is 112. The guessed point is $g = s_{65}$, the first MITM attack meets at s_{46}, and the second one meets at s_{73}. The attack steps are as follows.

1. Compute the 2 bits of $s_{46} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0\ldots77}$, and save all computation results in a table T_1 indexed by k_{f_1}.
2. Compute $s'_{73} = D_2(s_{112})$ by using every possible $k_{b_2} = k_{146\ldots222}$, and compute $\kappa_{130\ldots143}$ by treating $k_{144} = k_{145} = 0$. Save $k_{146\ldots222}$'s into a table T'_2 indexed by s'_{72} and $\kappa_{130\ldots143}$.
3. For each guess of $g = s_{65}$:
   (a) Compute the 2 bits of $s'_{46} = D_1(g)$ by using every possible $k_{b_1} = k_{114\ldots129}$. Store all k_{b_1}'s in a table T'_1 indexed by values of the 2 bits, and each entry is a sub-set containing certain k_{b_1}'s. After this step, T_1 and T'_1 together form a product set.
   (b) Compute $s_{73} = E_2(g)$ for each $k_{f_2} = k_{130\ldots145}$, calculate $\kappa_{130\ldots143}$ by subtracting k_{144} and k_{145} from $k_{130\ldots143}$, and find the matching $k_{146\ldots222}$ in T'_2. Next, based on the knowledge of $k_{144\ldots223}$, compute the sub-key pair of k_{f_1} and k_{b_1}, and check whether the pair is 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.

5 Further Optimization Methods

There are still many techniques that may help us reduce the attacks' complexities and reach more rounds.

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/decryptions. Consider the detailed steps of the partial matching used in the 2D-MITM attack on KATAN32 (see Sec 3.1.2). One of the two bits used for matching in s'_{87} has already been obtained 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 that. 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 the attack procedures very complicated to explain. In addition, in order to make 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, 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 reduce time and memory complexities, or extend attacks to more rounds.

The paper [19] 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 as a linear variable in intermediate states after a few rounds. Thus, if we look for possible matches of $s_i \oplus k_j$ instead of s_i, this bit information can still be used for matching. This technique may help us extend our attacks to more rounds.

6 Applications to KATAN with Less Rounds

Since the MD-MITM attacks in Section 3 and 4 focus on increasing the maximum numbers of attacked rounds and their time complexities are close to exhaustive search's, it is still questionable that whether this new kind of multidimensional approach can be practical enough to improve existing attacks on ciphers with less rounds. In this section, we will demonstrate how to apply MD-MITM attacks to reduced-round KATAN, and obtain less time complexities than the attacks in [1,14].

6.1 New Attack on 115-Round KATAN32

We will show how to attack KATAN32 with exactly the same number of rounds as in the paper [1], i.e. 115 rounds. For simplicity of description, we do not use any advanced optimization methods, such as partial matching.

This reduced-round attack is based on the one mentioned in Section 3.1.1, and our idea is to further reduce the time complexities by iterating less sub-key bits. The 115-round KATAN32 is segmented by s_{55}, and the MITM part for the first sub-cipher meets at s_{35} and the second MITM part meets at s_{79}. The attack procedure is also similar to previous attacks, and can be generally described as follows.

1. $k_{0...69}$ and $k_{70...109}$ in the first MITM phase forms a product set.
2. The second MITM phase of $k_{110...157}$ and $k_{158...229}$ yields a consecutive 80-bit sub-key $k_{150...229}$. Then we recheck $k_{150...229}$ in the product set constructed in the first MITM phase.
3. Passed keys are further examined by using additional pairs of plaintexts and ciphertexts.

The total time complexity of this new attack is

$$2^{70} \cdot \frac{35}{115} + 2^{72} \cdot \frac{36}{115} + 2^{32} \cdot \left(2^{40} \cdot \frac{20}{115} + 2^{48} \cdot \frac{24}{115}\right) + 2^{80-32} \approx 2^{77.75},$$

which is less than the time complexity of the attack in [1], i.e. 2^{79}. The memory complexity for this attack is $2^{70} + 2^{72} + 2^{40} \approx 2^{72.32}$.

6.2 New Attack on 100-Round KATAN48

Similar to the attack on 115-round KATAN32, we can construct a simple 2D-MITM attack on 100-round KATAN48. The guessed state is s_{48}. The first MITM part meets at s_{35} and the second one meets at s_{64}. $k_{0\ldots69}$ and $k_{70\ldots95}$ in the first MITM phase can form a product set, and $k_{120\ldots199}$ derived from the second MITM phase is checked again in the product set.

The overall time complexity is

$$2^{70}\cdot\frac{35}{100}+2^{72}\cdot\frac{36}{100}+2^{48}\cdot\left(2^{26}\cdot\frac{13}{100}+2^{32}\cdot\frac{16}{100}\right)+2^{80-48}\approx 2^{77.37}.$$

Although this simple attack may be further optimized, it has less time complexity than the attack on KATAN48 in [14]. The memory complexity of this new attack is $2^{70}+2^{72}+2^{26}\approx 2^{72.32}$, which is also less than the one in [14].

6.3 Discussions

A simple MD-MITM attack without any optimization will not get much advantage over traditional MITM attacks on KATAN64 with 94 rounds (the maximum number of rounds attacked in [14]), since its block size, i.e. 64 bits, is close to the key size, 80 bits. It is feasible to get lower time complexity by adopting partial matching as used on KATAN64 in Section 3.1.2. But as we already demonstrate the power of MD-MITM attacks on 115-round KATAN32 and 100-round KATAN48, and our primary goal is to extend attacks on KATAN to as many rounds as possible, we will leave the potential work on 94-round KATAN64 to interested readers.

One trick in our attacks, including the ones in Section 3 and 4, is to first derive a consecutive sub-key for later use, such as $k_{151\ldots230}$ used in the new attack on 115-round KATAN32. This is the most time-consuming step during the whole attack, and limits the lower bounds of time complexities for possible attacks. But since the computational cost of this step is low compared to a complete encryption/decryption of the cipher, the overall complexities of our attacks could be better than exhaustive search. However, we do not rule out the possibility of other ways to carry out MD-MITM attacks that may improve upon our methods.

The attacks on 115-round KATAN32 and 110-round KATAN48 proposed in this section demonstrate that this MD-MITM method not only can increase the numbers of possibly attacked rounds, but also could be used to improve the time and memory efficiencies of attacks on reduced-round ciphers. Compared to traditional MITM attacks that may need to iterate most of their attacking steps, MD-MITM approach can save computational consumption in certain phases, such as the first forward and last backward phases. This might also be one of the reasons why traditional MITM attacks cannot reach the same numbers of attacked rounds of KATAN as MD-MITM attacks.

7 Concluding Remarks

In this paper, we investigate a cryptanalysis framework called multidimensional meet-in-the-middle attack, which is applicable to lightweight ciphers with simple key scheduling algorithms and block sizes smaller than master key sizes. Refined analysis and attacks are presented on the block cipher family KATAN32/48/64 for demonstration. Our new attacks

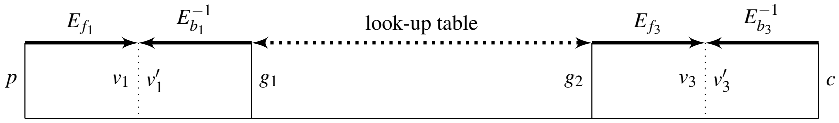

Fig. 5 Multidimensional MITM attacks with look-up tables.

reach many more rounds than existing attacks, and can also be more efficient than existing ones when being applied to KATAN with a smaller number of rounds.

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 approach to construct the loop-up table in Fig. 5, not only limited to MITM methods.

We may also analyze other lightweight ciphers as a future work. Consider KATAN's sibling block cipher family KTANTAN32/48/64 [5], which has the same round functions as KATAN but a simpler key scheduling algorithm. There already exist several attacks on full KTANTAN [19,4]. By refining our matching techniques, we may be able to construct more efficient attacks on KTANTAN.

Acknowledgement

The authors would like to thank Claude Carlet, Itai Dinur, Xinxin Fan, Yin Tan, and anonymous reviewers for helpful comments. This work is supported by NSERC Discovery Grant and ORF Grant.

References

1. M. R. Albrecht and G. Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.
2. K. Aoki and Y. Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.
3. A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
4. A. Bogdanov and C. Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.
5. C. D. Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.
6. N. T. Courtois. Algebraic complexity reduction and cryptanalysis of GOST, 2011. `http://www.nicolascourtois.com/papers/gostac11.pdf`.
7. W. Diffie and M. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.
8. I. Dinur, O. Dunkelman, N. Keller, and A. Shamir. 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*, pages 719–740. Springer, 2012.

9. I. Dinur, O. Dunkelman, and A. Shamir. Improved attacks on full GOST. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 9–28. Springer, 2012.
10. D. W. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith. The Hummingbird-2 lightweight authenticated encryption algorithm. In A. Juels and C. Paar, editors, *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 19–31. Springer, 2011.
11. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED block cipher. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
12. T. Isobe. A single-key attack on the full GOST block cipher. In A. Joux, editor, *Fast Software Encryption*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.
13. T. Isobe and K. Shibutani. Improved all-subkeys recovery attacks on FOX, KATAN and SHACAL-2 block ciphers. To appear at FSE 2014.
14. T. Isobe and K. Shibutani. All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2012.
15. D. Khovratovich, G. Leurent, and C. Rechberger. Narrow-bicliques: Cryptanalysis of full IDEA. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. Springer Berlin / Heidelberg, 2012.
16. L. R. Knudsen, G. Leander, A. Poschmann, and M. J. B. Robshaw. PRINTcipher: A block cipher for IC-Printing. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.
17. Y. Luo, Q. Chai, G. Gong, and X. Lai. A lightweight stream cipher WG-7 for RFID encryption and authentication. In *GLOBECOM*, pages 1–6. IEEE, 2010.
18. Y. Sasaki and K. Aoki. Finding preimages in full MD5 faster than exhaustive search. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.
19. L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, and S. Ling. Improved meet-in-the-middle cryptanalysis of KTANTAN (poster). In U. Parampalli and P. Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 433–438. Springer, 2011.

Appendix A Parameters for the KATAN Family of Block Ciphers

The parameters for the nonlinear functions (2) 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

Appendix B Partial Matching Details of the Attacks on KATAN48/64

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

```
Rd.   a b  L1                   L2
second backward phase
92    0 0  0000000000000000000  00000000000000000000000000000
91    1 1  0000000000000000011  00000000000000000000000000011
90    0 0  0000000000000001101  00000000000000000000000001100
89    0 0  0000000000000110110  00000000000000000000000110000
88    0 0  0000000000011011011  00000000000000000000011000001
87    0 0  0000000001101101101  00000000000000000001100000111
```

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

```
86    0 0   0000000110110110111  00000000000000000110000011110
85    0 0   0000011011011011111  00000000000000011000001111001
84    0 0   0001101101101111111  00000000000001100000111100111
83    0 0   0110110110111111111  00000000000110000011110011111
82    1 0   1011011011111111111  00000000011000001111001111111
81    0 1   1101101111111111111  00000001100000111100111111111
80    0 0   0110111111111111111  00000110000011110011111111111
79    0 0   1011111111111111111  00011000001111001111111111111
78    0 0   1111111111111111111  01100000111100111111111111111
77    0 0   1111111111111111111  10000011110011111111111111111
76    1 1   1111111111111111111  00001111001111111111111111111
75    0 0   1111111111111111111  00111100111111111111111111111
74    0 0   1111111111111111111  11110011111111111111111111111
73    0 0   1111111111111111111  11001111111111111111111111111
72    1 1   1111111111111111111  00111111111111111111111111111
second forward phase
71    0 0   0000000000000000000  00000000000000000000000000000
matching
2 bits      1111111111111111111  00111111111111111111111111111
```

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

```
Rd.   a b   L1                         L2
first forward phase
39    0 0   0000000000000000000000000  000000000000000000000000000000000000000
40    1 1   1110000000000000000000000  111000000000000000000000000000000000000
41    0 0   0001110000000000000000000  000111000000000000000000000000000000000
42    0 0   0000001110000000000000000  000000111000000000000000000000000000000
43    0 0   1100000001110000000000000  000000000111000000000000000000000000000
44    0 0   0011100000001110000000000  111000000000111000000000000000000000000
45    0 0   1110011100000001110000000  110111000000000111000000000000000000000
46    0 1   1111110011100000001110000  101110111000000000111000000000000000000
first backward phase
58    1 1   0000000000000000000000000  000000000000000000000000000000000000000
57    1 1   0000000000000000000000111  000000000000000000000000000000000000111
56    1 1   0000000000000000000111111  000000000000000000000000000000000111111
55    1 1   0000000000000000111111111  000000000000000000000000000000111111111
54    1 1   0000000000000111111111111  000000000000000000000000000111111111111
53    1 1   0000000000111111111111111  000000000000000000000000111111111111111
52    1 1   0000000111111111111111111  000000000000000000000111111111111111111
51    1 1   0000111111111111111111111  000000000000000000111111111111111111111
50    1 1   0111111111111111111111111  000000000000000111111111111111111111111
```

```
49   1 1  1111111111111111111111111  000000000000111111111111111111111111111
48   1 1  1111111111111111111111111  000000000111111111111111111111111111111
47   1 1  1111111111111111111111111  000000111111111111111111111111111111111
matching
2 bits    1111111111111111111111111  101110111111111111111111111111111111111
```