Improved Meet-in-the-Middle Cryptanalysis of KTANTAN

Lei Wei[1], Christian Rechberger[2], Jian Guo[3], Hongjun Wu[1], Huaxiong Wang[1], and San Ling[1]

[1] Division of Mathematical Sciences,
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
{weil0005,wuhj,hxwang,lingsan}@ntu.edu.sg
[2] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
christian.rechberger@groestl.info
[3] Institute for Infocomm Research, A*STAR, Singapore.
ntu.guo@gmail.com

Abstract. We revisit meet-in-the-middle attacks on block ciphers and recent developments in meet-in-the-middle preimage attacks on hash functions. Despite the presence of a secret key in the block cipher case, we identify techniques that can also be mounted on block ciphers, thus allowing us to improve the cryptanalysis of the block cipher KTANTAN family. The first and major contribution is that we spot errors in previous cryptanalysis, secondly we improve upon the corrected results. Especially, the technique *indirect-partial-matching* can be used to increase the number of matched bits significantly, as exemplified by our attacks. To the best of our knowledge, this is the first time that a *splice-and-cut* meet-in-the-middle attack is applied to block ciphers. When the splitting point is close to the start or the end of the cipher, the attack remains to be at very low data complexity. The secret key of the full cipher can be recovered faster than exhaustive search for all three block sizes in the KTANTAN family. The attack on KTANTAN32 works with a time complexity $2^{72.9}$ in terms of full round encryptions. The attack has a time complexity of $2^{73.8}$ and $2^{74.4}$ on KTANTAN48 and KTANTAN64, respectively. Moreover, all the three attacks work with 4 chosen ciphertexts only. These results compare favourably with the factor 2 speed-up over brute force obtained in earlier work[4], and hence these attacks are the best cryptanalysis results so far.

Keywords: block cipher, hash function, meet-in-the-middle attack, KTANTAN, indirect-partial-matching, splice-and-cut

1 Introduction

Low-end computing devices such as RFID tags are usually characterized by extremely tight cost and power consumption requirements. The needs of cryptography on such devices have been increasing with the growing pervasiveness and mass deployment of these devices. There are several block cipher primitives proposed in recent years, targeting very small footprint and reduced power consumption, in particular DESL [22], PRESENT [4], HIGHT [19], mCrypton [23], the KATAN/KTANTAN family [7] and PRINTcipher [21] which is proposed as a cryptography solution for IC-printing.

In this paper, we examine several techniques developed on meet-in-the-middle preimage attacks of hash functions and find that they are still effective when applied to block ciphers. With a circular configuration, we arrive at a more general framework compared to [6]. As an example we apply the attack to the KTANTAN family, the key of the full cipher can be found in time complexity faster than exhaustive search by an order of $2^{7.1}$, $2^{6.2}$ and $2^{5.6}$ for block size 32, 48 and 64. Moreover, these attacks require only 4 chosen ciphertexts.

Organization. In Section 2, we review the results and techniques developed in meet-in-the-middle attacks on both block ciphers and hash functions. In Section 3, we briefly introduce the

[4] This refers to [6], which was later updated to [5].

design of the KTANTAN family. In Section 4, we introduce earlier cryptanalysis in [6] followed by our observations and experiment results. In Section 5 we extend the meet-in-the-middle attack with the hash techniques in Section 2 to achieve the best cryptanalysis results. Section 6 concludes the paper with discussions and possible open problems.

2 Developments in MITM Attacks

Meet-in-the-middle (MITM) is a generic cryptanalytic approach originally developed from cryptanalysis of block ciphers. Early development starts with cryptanalysis of DES, which dates back to 1977. Similar ideas have been applied to round-reduced variants of AES in last 12 years. More recently, these techniques are found to be quite useful in preimages attacks of hash functions. We will review these results and techniques developed on DES, AES and hash functions in the following Section 2.1, 2.2 and 2.4, respectively.

2.1 Basic MITM Attacks on DES

DES is a block cipher standardized as FIPS 46 in 1976. It has a block size of 64 and key size 56. In an early cryptanalysis by Diffie and Hellman [13] in 1977, a meet-in-the-middle attack was proposed to examine the security of a cascade of the encryption function $E_K(\cdot)$ with two independent 56-bit keys K_1 and K_2. A ciphertext C is computed as $E_{K_2}(E_{K_1}(P))$ from plaintext P. Given text pair (P, C), an intermediate value M can be computed as both $E_{K_1}(P)$ and $E^{-1}_{K_2}(C)$, with parallel guesses of K_1 and K_2. By first storing values of M computed from P with all values of K_1, compute M' with guesses of K_2 to find match with the stored M values, the correct (K_1, K_2) pair certainly gives a match and is expected to be revealed by excluding the false positives with additional text pairs. The attack is able to recover 112-bit key in at most 2^{57} rather than 2^{113} in number of full DES encryptions.

In practice it is rare that key materials are completely separated into two halves of the cipher, as in the case of doubly cascaded DES. Meet-in-the-middle remains effective even in less desirable scenarios. In 1985, Chaum and Evertse published several meet-in-the-middle attacks against reduced DES [8]. The idea is to find "bits in the middle" that are independent of certain key bits when computed from both ends. For an R-round case with encryption function $E_K(\cdot)$, let the $G_K(\cdot)$ be the first S rounds and $H_K(\cdot)$ be the rest $R - S$ rounds, we have $E_K(\cdot) = H_K \circ G_K(\cdot)$. Given text pair (P, C), we can compute the intermediate values $M := G_K(P)$ and $M' := H^{-1}_K(C)$ for a key guess K. The attack is effective if there is $\tilde{K} \subsetneq K$ and for some nonzero mask b such that $b \cdot G_{\tilde{K}}(P) = b \cdot H^{-1}_{\tilde{K}}(C)$ for all (P, C) pairs, where $\cdot$ is the bit-wise logic AND. The $wt(b)$-bit match can be used to filter $2^{-wt(b)}$ ratio of wrong keys during the MITM attack, $wt(\cdot)$ computes the weight of the binary vector b. The attack covers at most 7 rounds (round 2-8) of DES faster than exhaustive search. Further work due to [15] on DES reduced to $4, 5, 6$ rounds has involed novel methods to reduce the time complexity.

2.2 MITM Attacks on AES and Other Block Ciphers

MITM attacks against AES are a bit different from the techniques described in last subsection. It aims to find some special properties of $M := G_K(P)$, such as the *balanced property*, and then try to decrypt (*i.e.,* compute $M' := H^{-1}_K(C)$) the second half to match M' with the special properties so that wrong key guesses can be filtered out. This type of attacks on AES starts with a 6-round result in the Rijndael submission [9], then has been improved to 7/8-round results in a series of work [10, 12, 14, 16]. More dedicated attacks with meet-in-the-middle technique can be found in the practical attack to KeeLoq [20] and cryptanalysis of reduced round IDEA [11].

2.3 More Recent MITM Attacks

In [6] the framework is made clearer with different notions as compared with the basic idea in [8]. Some key bits of K are not used in the first S_1 rounds, *i.e.,* $G_K = G_{K_1}$ with $K_1 \subsetneq K$ being the key bits essentially used in the first S_1 rounds. Similarly, $H_K = H_{K_2}$ with $K_2 \subsetneq K$ for the last S_2 rounds. In addition, the matching is done in a separate partial matching phase of the middle $R - S_1 - S_2$ rounds transformation $I(\cdot)$ from two fully determined states $M := G_{K_1}(P)$ and $M' := H^{-1}(K_2)$, *i.e.,* $E_K(\cdot) = H_{K_2} \circ I \circ G_{K_1}(\cdot)$. The formalization allows better understanding of the attack and may breed powerful extensions. We will discuss the attack details in Section 4, which is the first attack that applies to the full KTANTAN32. Similar MITM attacks have been applied to block ciphers including reduced GOST [26] and XTEA [27] recently.

2.4 Developments of MITM Preimage Attacks on Hash Functions

In 2008, Aoki and Sasaki noticed that the MITM attacks could be applied to hash functions, to find (second) preimages faster than brute-force. The attacks have successfully broken the one-wayness of several designs in the MD4 hash family including MD5 [25], round-reduced SHA-0 and SHA-1 [2], round-reduced SHA-2 [1], and also some other Davies-Meyer hash constructions, *e.g.,* Tiger [17], reduced HAS-160 [18] and HAVAL [24][5]. Besides the basic techniques, several new techniques have been developed in these hash attacks, namely, *splice-and-cut*, *initial structure* (IS), *partial-matching* (PM), and *indirect-partial-matching*. We describe them here in detail as in Fig. 1, and later in the paper apply to KTANTAN family to obtain the best cryptanalytic results. A detailed review of MITM preimage attacks on hash functions can also be found in [17].

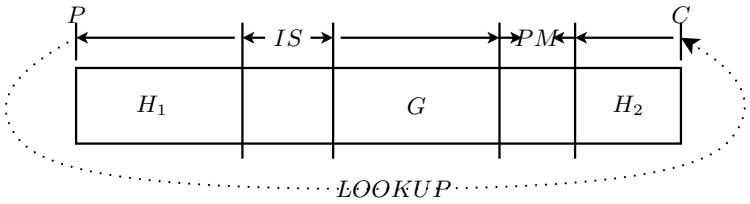

Fig. 1. A general setup for MITM attacks

Splice-and-Cut To prevent (second) preimage attacks for narrow-pipe (*i.e.,* the state size equals to digest size) hash designs, feedforward is usually required. Davies-Meyer is among the most popular ways to construct a compression function from a block cipher, *i.e.,* $h' = E_m(h) \oplus h$, where $\oplus h$ is called feedforward and E is a block cipher keyed by message m. In contrast to the MITM attacks against block ciphers, *splice-and-cut* makes use of the feedforward, and split the block cipher $E(\cdot)$ into three sub-ciphers, *e.g.,* $E(h) = H_2 \circ G \circ H_1(h)$. Let $h_{inter} = H_1(h)$, we can refine the expression to $H_2^{-1}(H_1^{-1}(h_{inter}) \oplus h') = G(h_{inter})$. With the information of h and h', one can compute the output $h \oplus h'$ of the block cipher, this is called *splice*, which connects the input and output of E. The attack starts with the intermediate point h_{inter} of the E, and *cut* the compression function into two parts: $G(\cdot)$ and $H_2^{-1}(H_1^{-1}(\cdot) \oplus h')$, then carry out the MITM attacks (note in a preimage attack against hash functions, h' is given as the target, hence can be treated as a constant and values of h_{inter} can be chosen freely). This enables the attacker to choose the splitting point (position of h_{inter}) wherever he wants. Hence it allows to search for globally better set of neutral bits which might lead to a lower attack complexity.

[5] We only cite the best attacks on each hash function here.

In case of block ciphers, such feedforward does not exist. However, one can still build a *virtual* feedforward by making use of a lookup table between plaintext and ciphertext. Instead of using $H_1^{-1}(h_{inter}) \oplus h'$ as the output of E, we use $LOOKUP(H_1^{-1}(h_{inter}))$, where $LOOKUP$ outputs a ciphertext given the plaintext $H_1^{-1}(h_{inter})$ under the correct key. Hence the MITM attack can be carried out from the two sides of the equation $H_2^{-1}(LOOKUP(H_1^{-1}(\cdot))) = G(\cdot)$. The original MITM attacks can be viewed as a special case in this generalized setting with H_1 being the identity, *i.e.,* H_1 covers 0 round with h_{inter} being the plaintext.

The table $LOOKUP$ can be constructed by computing from a randomly chosen h_{inter}. The set of possible $H_1^{-1}(h_{inter})$ for varying keys are the chosen plaintexts to index the ciphertexts accordingly. The entire codebook is necessary when H_1 involves more secret key bits than the block size. If much less entries are used in the table, it can be further compressed to a dense table to reduce memory requirement. The data complexity of the attack is the number of entries in this table. More details are in Section 5.

Initial Structure In a hash attack, one can choose the value of h_{inter} freely. It is noted that, under certain conditions, some neighbouring key bits can be swapped, and yet leave the computation result unchanged. With help of this, some steps near h_{inter} can be skipped as demonstrated in Fig. 1, and smaller size of K_1 and K_2 can be found for the forward computation G_{K_1} and backward H_{K_2}, hence a reduced time complexity is possible.

Partial-Matching Let $K_c := K_1 \cap K_2$, *i.e.,* the key bits used in both computing forward for M and backward for M' from the same h_{inter}. It is noted that it is not necessary to match M and M' fully at the early stage of the attacks. For $2^{|K_1|-|K_c|}$ candidates of M, and $2^{|K_2|-|K_c|}$ candidates of M', it is expected to have $2^{|K_1|+|K_2|-2|K_c|-m}$ pairs left if only m bits are checked and matched. Then one can further check the candidates left if the rest of bits in M and M' are also matched. This additional step takes about $2^{|K_1|+|K_2|-2|K_c|-m}$ full encryptions. Actually it is much less due to the fact that the re-check needs only to repeat the portion of PM as shown in Fig. 1. Let α denotes the percentage of steps covered by PM over that of the full cipher. The re-check will not dominate the overall complexity when

$$(|K_1| + |K_2| - 2|K_c| - m) + \log_2(\alpha) < \max(|K_1| - |K_c|, |K_2| - |K_c|).$$

Indirect-Partial-Matching The indirect partial matching is a way to extend the partial matching for more steps. Usually partial matching starts only when key bits in $K_2 \setminus K_c$ appear after the end of G (otherwise, one can extend G for more steps). Similarly for the other side, *i.e.,* key bits in $K_1 \setminus K_c$ appear just before H_2. If one can find several bits in the IS portion such that it can be expressed as $G_{K_1} + \phi_{K_2}$ from one direction, and $H_{K_2} + \mu_{K_1}$ from the other direction. Instead of computing the actual values and doing the match directly, one can compute $G_{K_1} - \mu_{K_1}$ and $H_{K_2} - \phi_{K_2}$ independently from two directions to check for match. The power of this technique will be demonstrated in our results in Section 5.

There are more techniques such as partial-fixing, precomputed-initial-structure, probabilistic-partial-matching developed in these hash attacks, however they are not much related to our results.

3 The KTANTAN Family of Block Ciphers

KATAN and KTANTAN are two flavors of a family of hardware oriented block ciphers proposed at CHES'09 [7]. The design is aiming at encryption needs on low-end devices such as RFID

tags hence the cipher is targeting low gate count and reduced power consumption, besides an 80-bit security level supported by a large security margin on differential cryptanalysis, linear cryptanalysis and other attacks.

Both KATAN and KTANTAN have three variants with block sizes of 32-bit, 48-bit and 64-bit and each of them takes 80-bit user key. In addition, KATAN and KTANTAN share the same data path specification, including the round transformation and round constants. The data path is best illustrated with Fig. 2 cited from [7]. The round transformation is repeated for 254 rounds with a sequence of values of IR_i for $i = 1 \ldots 254$.

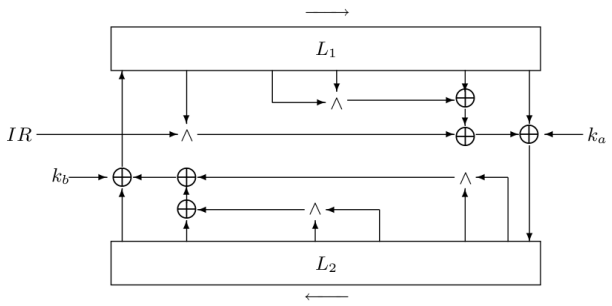

Fig. 2. Illustration of KATAN/KTANTAN round transformation

The cipher state consists two parts L_1 and L_2 with different lengths. During the encryption, the plaintext is loaded into the two registers for L_1 and L_2 in little-endian. In each round of KTANTAN32, L_1 and L_2 are shifted to the left by 1 position with their least significant bits filled by results of $f_{b,r}(\cdot)$ and $f_{a,r}(\cdot)$, respectively. $f_{a,r}$ and $f_{b,r}$ are the only non-linear functions in the round transformation, defined as

$$f_{a,r}(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR_r) \oplus k_{a,r} \tag{1}$$

$$f_{b,r}(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_{b,r} \tag{2}$$

where $k_{a,r}$ and $k_{b,r}$ are 2 bits subkey of round r and IR_r is the round r constant. For the case of KTANTAN48 and KTANTAN64, the round transformation is repeated 2 and 3 times, respectively. The specification of version specific constants can be found in Table 1 and IR in Table 2.

Table 1. Parameters defined for the KATAN/KTANTAN family of ciphers

Blocksize	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
32	13	19	12	7	8	5	3	18	7	12	10	8	3
48	19	29	18	12	15	7	6	28	19	21	13	15	6
64	25	39	24	15	20	11	9	38	25	33	21	14	9

Differences in round transformation are introduced first by IR, a 254-bit binary string termed the *irregular update sequence*. In addition, two bits k_a and k_b are generated from the 80-bit user key for reach round. How the subkeys are generated is the only difference between KATAN and KTANTAN. Here we discuss the key schedule of KTANTAN. Two bits of the 80-bit $K = k_{79}k_{78} \ldots k_1k_0$ are selected each round, by the state of an 8-bit LFSR defined by the

Table 2. Round constant: the irregular update sequence

1111111000 1101010101 1110110011 0010100100 0100011000 1111000010
0001010000 0111110011 1111010100 0101010011 0000110011 1011111011
1010010101 1010011100 1101100010 1110110111 1001011011 0101110010
0100110100 0111000100 1111010000 1110101100 0001011001 0000001101
1100000001 0010

feedback polynomial $x^8 + x^7 + x^5 + x^3 + 1$. With an initial value 0xFF, it is clocked 254 times. In each round the state of the LFSR is used to select two bits from K.

First divide K into five 16-bit words, let $K = w_4||w_3||w_2||w_1||w_0$, where the LSB of w_0 is the LSB of K and the MSB of w_4 is the MSB of K. Let $T_7T_6 \cdots T_1T_0$ be the state of LFSR at a particular round r, then let $a_i = \text{MUX16to1}(w_i, T_7T_6T_5T_4)$ where MUX16to1 is a multiplexer that outputs the value of a bit in w_i with position specified by $T_7T_6T_5T_4$. After that, the subkey bits $k_{a,r}$ and $k_{b,r}$ are selected as

$$k_{a,r} = \bar{T_3} \cdot \bar{T_2} \cdot (a_0) \oplus (T_3 \vee T_2) \cdot \text{MUX4to1}(a_4a_3a_2a_1, T_1T_0)$$

$$k_{b,r} = \bar{T_3} \cdot T_2 \cdot (a_4) \oplus (T_3 \vee \bar{T_2}) \cdot \text{MUX4to1}(a_3a_2a_1a_0, \bar{T_1}\bar{T_0})$$

where MUX4to1 is a multiplexer that selects 1 bit from 4, similar as MUX16to1. In fact, when the LFSR is clocked each round, the MSB is taken to form the irregular update sequence. The reference IR may be useful to verify the implementation of the key schedule. It is noted that the reference implementation has been updated several times, we list the entire key schedule in Appendix A to avoid ambiguity, in accord with implementation [3].

4 The Previous Meet-in-the-Middle Attack

Here we briefly describe the attack in [6]. The attack manages to recover the secret key with very low data complexity, as required by the *unicity distance* in the known plaintext setting. The attack to KTANTAN32 runs at a time complexity of 2^{79}, although arguably marginal, is the first key-recovery attack applicable to the full 254 rounds faster than brute force. For block size b of 48 and 64, the attack manages to break 251 and 248 rounds respectively.

The main idea of this attack is to cut the R-round KTANTAN-b cipher into three parts for some $\alpha, \beta < R$, *i.e.,* the first α rounds as the *forward* phase, the last β rounds as the *backward* phase and the middle $R - \alpha - \beta$ rounds as the *partial matching (MITM)* phase. Let x_i be the state after round i, for $0 \leq i \leq 254$. Let $\varphi_{i,j}$ be the transformation from round i to round j (inclusive) of the R-round KTANTAN-b, under an unknown fixed key $K := k_{79}k_{78} \ldots k_1k_0$. It is found that the key bits in $A_1 := \{k_{15}, k_{79}\}$ are neutral to $H := \varphi_{254-\beta+1,254}$ (*i.e.,* the last β rounds) for $1 \leq \beta \leq 118$, and $A_2 := \{k_5, k_{37}, k_{69}\}$ neutral to $G := \varphi_{1,\alpha}$ (*i.e.,* the first α rounds) for $1 \leq \alpha \leq 105$. The rest of the bits $A_0 := \{k_0, k_1, \ldots, k_{78}, k_{79}\} \setminus (A_1 \bigcup A_2)$ are used in both the forward and the backward phases. The attack to KTANTAN32 proceeds as described in Section 2 with a text pair (P, C).

For each guess of key bits in A_0:

1. For each guess of A_1, compute $M := G(P) = \varphi_{1,105}(P)$. Then, 3-bit of x_{128} can be computed from M and used as an index to store value of A_1 in a table.
2. For each guess of A_2, compute $M' := H^{-1}(C) = \varphi^{-1}_{137,254}(C)$. Compute the same 3-bit of x_{128} from M'. If the value is the index to some stored A_1, do the *key testing.*

Key testing: Test whether $M' = \varphi_{106,136}(M)$ for K, *i.e.,* current guessed values for bits in A_0, A_1 and A_2. If K passes, try additional pairs one by one. Continue the search if K is rejected. If K manages to survive all pairs, we conclude that it is the correct key with probability close to 1.

Data Complexity: Roughly $\lceil 80/b \rceil$ text pairs are necessary to identify the correct key for block size b. (*i.e.,* each pair has a filtering power of 2^{-b}). Each time the key passes the test of one pair, an additional pair is used, until the key is rejected or all text pairs exhausted - we conclude that K is the correct guess.

Time Complexity: The claimed complexity is at around $2^{|A_0|}(2^{|A_1|}+2^{|A_2|})+2^{80-3} = 2^{79}$ basic operations. Considering that a more accurate calculation will better reflect the attack at the marginal value, the number of full block encryptions that the attack works at is

$$2^{|A_0|}(2^{|A_1|} \cdot \alpha/R + 2^{|A_2|} \cdot \beta/R + 2^{|A_1|+|A_2|-m}(R-\alpha-\beta)/R) \doteq 2^{77.6}.$$

4.1 New Experimental Observations on the Attack

We have implemented the family of KTANTAN and examined the attack by Bogdanov and Rechberger in [6], with respect to both the cipher design paper [7] and its reference implementation [3]. We managed to find different sets of neutral key bits for the KTANTAN key schedule, these neutral key bits lead to the first non-marginal attack of the cipher family for the full rounds of all the three block sizes. The new set of neutral key bits are presented in Table 3. The first example of the comparison to the B-R attack on KTANTAN32 would be that the set of key bits A_2 neutral to $\varphi_{1,105}$ is $\{13, 27, 32, 39, 44, 59, 61, 66, 75\}$, and $A_1 = \{3, 20, 41, 47, 63, 74\}$ neutral to $\varphi_{137,254}$ while at most 3 bits can be matched in the middle[6]. We compare the attacks in Table 3.

Table 3. The B-R attack and our results

b	R	α	β	A_1	A_2	m	Time	Data	
32	254	105	118	15, 79	5, 37, 69	3	$2^{79.0}$	3 KP	[6]
48	251	107	112	11, 15, 75, 79	5, 69	1	$2^{79.7}$	2 KP	[6]
64	248	107	112	9, 73	5, 69	2	$2^{79.58}$	2 KP	[6]
32	254	111	122	3, 20, 41, 47, 63, 74	32, 39, 44, 61, 66, 75	12	$2^{73.88}$	3 KP	This paper
32	254	110	122	3, 20, 41, 47, 63, 74	27, 32, 39, 44, 59, 61, 66, 75	4	$2^{73.88}$	3 KP	This paper
32	254	109	122	3, 20, 41, 47, 63, 74	13, 27, 32, 39, 44, 59, 61, 66, 75	3	$2^{74.33}$	3 KP	This paper
48	254	123	122	3, 20, 41, 47, 63, 74	32, 44, 61, 66, 75	37	$2^{74.53}$	2 KP	This paper
48	254	111	121	3, 20, 41, 47, 63, 74	32, 39, 44, 61, 66, 75	4	$2^{73.97}$	2 KP	This paper
64	254	123	122	3, 20, 41, 47, 63, 74	32, 44, 61, 66, 75	44	$2^{74.53}$	2 KP	This paper

For an attack with m-bit matching value and b-bit block size we follow a more detailed calculation of the time complexity in number of full block encryptions (for $m > |A_1| + |A_2|$ the last term can be dropped):

$$2^{|A_0|}(2^{|A_1|} \cdot \alpha/R + 2^{|A_2|} \cdot \beta/R) + 2^{80-m} \cdot (R-\alpha-\beta)/R.$$

[6] This refers to the pre-proceeding version [6]. The authors updated in [5]. It is noted that the reference implementation had been updated several times, some confusion stemming from updated reference implementations seem to be the cause of the different attack results. Nevertheless, our attack applies to the cipher according to the design paper, and also in accord with the final version of the reference implementation retrieved.

4.2 Low Complexity Implementation of the Attack

The attack works at time complexity above 2^{70} which is still out of reach for nowadays computing power. Nevertheless, due to the very low data complexity at merely the unicity distance, it is feasible to launch the attack with just a handful of text pairs (*e.g.,* by eavesdropping a small amount of ciphertexts that correspond to known protocol headers). Moreover, in the cases that the secret keys are not derived with full 80-bit entropy, such an attack may become a real threat when time complexity becomes 2^6 to 2^7 times less due to the attack of this paper.

We implement such an attack scenario as a cryptanalytic game, one of the authors as the attacker tries recovering the complete secret key when a few plaintexts and their ciphertexts are given by another author, who then verifies whether the attack is successful as only he knows the real key. We assume that the attacker has already gained knowledge of part of the key bits in A_0. The attack works on the full 254-round KTANTAN32, with $\alpha = 111, \beta = 122, A_1 = \{3, 20, 41, 47, 63, 74\}, A_2 = \{32, 39, 44, 61, 66, 75\}$. By computing forward up to the end of round 126 and backward before round 127, there are 12 bits match. The details of the matching are given in the same notations used in [6]. *e.g.,* $k_1 = 1$ means the key bit is active (unknown) and $k_2 = 0$ means the key bit is inactive (known). An 1 in the state means the bit is affected by an active key bit, and 0 means it is not affected (*i.e.,* this state bit can be fully determined from previous state).

```
forward part: Neutral bits: 32 39 44 61 66 75

 forward R=112: k1=1, k2=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 1
 forward R=113: k1=0, k2=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 1 0
 forward R=114: k1=0, k2=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 1 0 0
 forward R=115: k1=0, k2=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 1 0 0 0
 forward R=116: k1=0, k2=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 1 0 0 0 1
 forward R=117: k1=0, k2=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 1 0 0 0 1 0
 forward R=118: k1=0, k2=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 1 0 0 0 1 0 1
 forward R=119: k1=0, k2=0,  0 0 0 0 0 1 0 0 0 1 0 0 0  0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0
 forward R=120: k1=0, k2=0,  0 0 0 0 1 0 0 0 1 0 0 0 1  0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1
 forward R=121: k1=0, k2=0,  0 0 0 1 0 0 0 1 0 0 0 1 1  0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1
 forward R=122: k1=0, k2=0,  0 0 1 0 0 0 1 0 0 0 1 1 1  0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1
 forward R=123: k1=0, k2=0,  0 1 0 0 0 1 0 0 0 1 1 1 1  0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0
 forward R=124: k1=1, k2=0,  1 0 0 0 1 0 0 0 1 1 1 1 1  0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 1
 forward R=125: k1=0, k2=0,  0 0 0 1 0 0 0 1 1 1 1 1 1  0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 1 1
 forward R=126: k1=0, k2=0,  0 0 1 0 0 0 1 1 1 1 1 1 1  0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 1 1 1

backward part: Neutral bits: 3 20 41 47 63 74

backward R=132: k1=0, k2=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
backward R=131: k1=1, k2=1,  1 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
backward R=130: k1=0, k2=0,  0 1 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
backward R=129: k1=0, k2=0,  0 0 1 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
backward R=128: k1=0, k2=0,  0 0 0 1 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
backward R=127: k1=0, k2=0,  1 0 0 0 1 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

Bit map of 12 matched bits:  1 0 1 0 1 0 1 1 1 1 1 1 1  0 0 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 1
```

The game works as follows:

1. First, player A chooses an 80-bit secret key and generates 3 text pairs. Then he sends the lower 40 bits of A_0 together with the text pairs to Player B. The 80-bit secret key in bit stream order looks as:

```
100*1101 01001010 0101*111 01010110 *111101* 1*01*0** ******** ******** ******** ********
```

where $*$ denotes unknown bit and 0/1 the known bit. The text pairs are (`0x1314B499`, `0xE995A2BE`), (`0x39AF5959`,`0xA7B2C77C`), and (`0xF5128319`, `0x1A54138F`).

2. Second, player B recovers the bits under $*$ with the following results: A_1 : `111110`, A_2 : `010100` and A_0 : `10011001110010111011100001000`, where each bit from low to high order corresponds to the masked bits in A_1, A_2 and A_0.

3. Lastly, player A acknowledges that the recovered bits are correct.

Note: The attack in Step 2 successfully recovers the 40-bits in 5 hours 34 seconds on a Quad-core HP xw4600 workstation of 2.40GHz. The implementation makes use of the reference bit-slice code which fits well in this attack as 2^6 instances of partial transformations are required on both the forward and backward phase, *i.e.,* for $\varphi_{1,111}$ and $\varphi^{-1}_{133,254}$ respectively. On each direction, the 64 bit-sliced instances are loaded with 64 guesses of A_1 and A_2 respectively. With 40 bits known, the estimated complexity 2^{74} is reduced to around 2^{34} full encryptions. The estimation matches the experiment well since the bit-sliced reference implementation encrypts 2^{26} text blocks in roughly 45 seconds on this workstation. As a comparison, recovering 40 bits by exhaustive search will take roughly half a month using the same computational resources.

5 More General MITM Attacks on KTANTAN Family

The attacks described in Section 4 are due to a weakness in the key schedule of the KTANTAN family, especially, the efficiency of the attacks depends crucially on how many neutral key bits can be found on the forward phase and the backward phase. Hence a natural question is, is there an optimal partitioning if we are allowed to relax some requirements? We employ the *splice-and-cut* technique described in Section 2.4, start by randomly choosing a value for the state at a middle round and compute both forward and backward to check how many bits can be matched by the *indirect-partial-matching* technique. The core idea of this setup is almost identical with the basic form, *i.e.,* the B-R attack. For m-bit match, under the correct key guess, the match and re-check followed will all pass while for a random wrong guess, the match will hold with probability 2^{-m}, *i.e.,* a ratio of 2^{-m} of the overall guesses will be filtered in the partial matching phase. This effectively reduces the amount of key testing below brute-force.

5.1 Indirect Partial Matching

We employ this technique due to an important observation on the mixing of key bits. In each round the transformation with $f_{r,a}$ and $f_{r,b}$ are repeated 1, 2 and 3 times for respective block size 32, 48 and 64. For each of the evaluations of (1) and (2), a single round key bit is mixed XOR-linearly into the LSB of L_1 or L_2, hence affecting the lowest 1 to 3 bits considering the shift(s). In the round that follows, very few bits of the state get involved in the nonlinear terms of (1) and (2) when computing $f_{r,a}$ and $f_{r,b}$. This means after the round key bits are mixed into the state, they will remain linear for quite a number of rounds. With this observation, the *indirect-partial-matching* technique is considered to try tracing more state bits during the partial matching. These additional state bits are those affected by active key bits, but just linearly affected.

5.2 Splice-and-Cut

Let x_i be the state after round i for $1 \leq i \leq 254$ then x_{254} is the ciphertext and denote the plaintext x_0. Let b_0 and b_1 be the round numbers of the start and the end of backward

computation, and f_0 and f_1 be that of the forward computation. The rounds between (exclusive) b_0 and f_0 are used for the *initial structure*. However in the case of KTANTAN, the search result shows that this technique is not particularly relevant to a better attack, therefore we are setting f_0 to $b_0 + 1$ to allow no initial structure. The indirect partial matching is performed between two fully determined states $M := x_{f_1}$ (after round f_1) and $M' := x_{b_1-1}$ (before round b_1). The search is done with the fact in mind, that a huge $LOOKUP$ table may be necessary to splice the two ends of cipher together as described in Section 2.4. Nevertheless, the block sizes of all three versions are less than the key size, in the worst case, the key recovery attack will have to use the entire codebook. Note that the attack is still strictly better than owning the codebook since it is able to recover the secret key.

From the search result shown in Table 4, in the optimal partition of KTANTAN32 and KTANTAN64 as well as a near-optimal partition of KTANTAN48, the backward phase and the forward phase are split before round 254. In total only 4 ciphertexts have to be computed from the chosen state x_{253} as $\varphi_{254,254}(x_{253})$ with all possible values of the round key $k_{71}k_7$. Hence the data complexity is at 4 chosen ciphertexts. For each iteration of the attack, we refer the active entry of the $LOOKUP$ table the *active pair*, and other entries in the table as *non-active* pairs. During the key testing, besides the reuse of the active pair, one or two additional non-active pairs can be used to reject false positives. In the optimal partition of KTANTAN48 that gives the least time complexity, the entire codebook is necessary since the chosen state x_{107} is in the middle of the cipher. In Fig. 3 below we illustrate the cases in which the splitting point is at the end of the cipher. With notations of previous sections, $G := \varphi_{1,f_1} \circ LOOKUP \circ \varphi_{f_0,254}$ and $H := \varphi_{b_1,b_0}$.

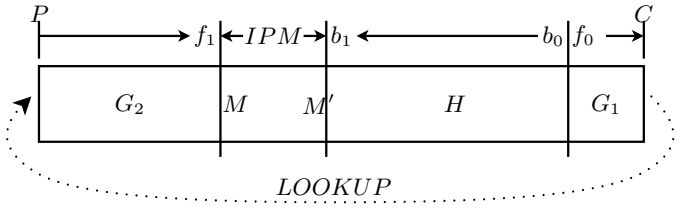

Fig. 3. Illustration of MITM attack with splice-and-cut

5.3 The Attack with Splice-and-Cut and Indirect-Partial-Matching

The first step is to construct the table $LOOKUP$ as to splice the two ends of cipher. It is done with the chosen ciphertext (C, P) pairs, or the entire codebook. For the attack illustrated by Fig. 3, the splitting point is at the end, we select a random value for x_{b_0} and compute $C := G_1(x_{b_0})$ for all the possible values of key bits involved in $G_1 := \varphi_{f_0,254}$. For each C find the corresponding plaintext P and add (C, P) to the table $LOOKUP$ with C as the index and P as the value. If the splitting point is in a middle round and we need the entire codebook, $LOOKUP$ is constructed by adding all the (C, P) pairs when $f_1 < f_0$ or all the (P, C) pairs when $b_0 < b_1$. We do not discuss this scenario since it is almost identical to the algorithm for the illustrated attack. Following the notations we use A_1 for the key bits neutral to the backward transformation $H^{-1} := \varphi^{-1}_{b_1,b_0}$ and A_2 for the key bits neutral to forward $G := G_2 \circ LOOKUP \circ G_1 = \varphi_{1,f_1} \circ LOOKUP \circ \varphi_{f_0,254}$.

Attack Algorithm: For each guess of key bits in A_0:

1. For each guess of A_1, compute $M := G(x_{b_0})$. Then, m-bit *partial matching signature* can be computed from M and used as an index to store value of A_1 in a table.

2. For each guess of A_2, compute $M' := H^{-1}(x_{b_0})$. Then compute the same m-bit partial matching signature from M'. If the value is the index to some stored A_1, do the *key testing*.

Indirect Partial Matching: The m-bit partial matching signature are m bits values at the position of the matched bits, computed as described in Section 2.4 on indirect partial matching. Searching of such signature requires to find state bits either independent of the active key bits in the IPM phase or dependent but only linearly affected. The technique significantly improved the number of bits that can be matched. This allows us to have more rounds for partial matching hence possibly more neutral bits. The detailed matching of the current optimal attack to KTANTAN32 can be found in Appendix B. To compute the matching signature, we take KTANTAN32 for example, the matching point in the optimal attack shown above is after round 115. To simplify the notations, let $x := x_{115}$ and $x[i]$ be the i-th bit of x, for $0 \leq i \leq 31$. For the forward part of the indirect partial matching, it is not hard to find the following key bits appear only as linear terms in corresponding state bits, k_{27} in $x[0]$, k_{13} in $x[1]$, k_{39} in $x[3]$, k_{59} in $x[4]$ and k_{39} in $x[22]$. For the backward part, similarly we have linear terms of k_{74} in $x[26]$, k_{74} in $x[21]$, k_{74} in $x[3]$ and k_{20} in $x[2]$. Hence the matching signature is $(x[26] - k_{74}, x[22] - k_{39}, x[21] - k_{74}, x[7], x[6], x[5], x[4] - k_{59}, x[3] - k_{39} - k_{74}, x[2] - k_{20}, x[1] - k_{13}, x[0] - k_{27})$ which can be computed from both sides without knowing the value for the active key bits from that side.

Key testing: Test whether $M' = \varphi_{f_1+1,b_1-1}(M)$ for K, *i.e.,* current guessed values for bits in A_0, A_1 and A_2. If K passes, try non-active pairs. Continue the search if K is rejected. If K manages to survive a number of pairs at the unicity distance, we conclude that it is the correct key with probability close to 1.

Table 4 shows the result of search that leads to better complexities.

Table 4. MITM attack with splice-and-cut and indirect-partial-matching techniques

b	R	b_1	b_0	f_0	f_1	A_1	A_2	m	Time	Data	
32	254	148	253	254	109	$13, 27, 32, 39, 44, 59, 61, 66, 75$	$3, 20, 41, 47, 63, 74$	11	$2^{72.93}$	4 CC	This paper
48	254	246	107	108	208	$0, 3, 32, 47, 60, 63, 64$	$13, 27, 39, 44, 59, 61$	9	$2^{73.43}$	2^{48} KP	This paper
48	254	150	253	254	111	$32, 39, 44, 61, 66, 75$	$3, 20, 41, 47, 63, 74$	15	$2^{73.77}$	4 CC	This paper
64	254	151	253	254	112	$32, 44, 61, 66, 75$	$3, 20, 41, 47, 63, 74$	54	$2^{74.38}$	4 CC	This paper

6 Conclusions

We presented new experimental observations that lead to improved cryptanalytic results for the KTANTAN family. In addition, we have implemented the attack to KTANTAN32 in a low complexity version. The result confirms the validity of such attacks and the reduction on time complexity. Moreover, the techniques developed from cryptanalysis of hash functions in the MITM framework are shown to be effective on analysis of block ciphers as well, despite the presence of a secret key, as we obtained the best cryptanalysis on KTANTAN family. In particular, the *splice-and-cut* technique allows searching of optimal partitioning on neutral bits and match. In conjunction with *indirect-partial-matching*, significantly more bits can be matched in our attacks, this helps to have more rounds covered by partial matching hence it becomes possible to have more neutral bits to improve the time complexity. Other techniques like *precomputed-initial-structure* do not contribute to our attack, but we expected them to be effective in analysis of other block ciphers in the MITM framework. It is yet open to examine the real potential of

MITM attacks in general on block cipher cryptanalysis if more powerful enhancements can be discovered. More dedicated matching techniques might be possible for particular block ciphers.

References

1. Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 578–597. Springer, 2009.
2. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 70–89. Springer, 2009.
3. Jean-Phillipe Aumasson, Miroslav Knezevic, and Orr Dunkelman. Bit-sliced Reference Implementation of KATAN/KTANTAN Family. `http://www.cs.technion.ac.il/~orrd/KATAN/katan.c`, 2010. Accessed on 6 August 2010.
4. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.
5. Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography*, pages 229–240, 2010.
6. Andrey Bogdanov and Christian Rechberger. Generalized Meet-in-the-Middle Attacks: Cryptanalysis of the Lightweight Block Cipher KTANTAN. 2010. In preproceedings of SAC, `http://homes.esat.kuleuven.be/~abogdano/talks/ktantan_sac10.pdf`.
7. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.
8. David Chaum and Jan-Hendrik Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *LNCS*, pages 192–211. Springer, 1985.
9. Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael. NIST AES proposal, 1998.
10. Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 116–126. Springer, 2008.
11. Hüseyin Demirci, Ali Aydin Selçuk, and Erkan Türe. A New Meet-in-the-Middle Attack on the IDEA Block Cipher. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 117–129. Springer, 2003.
12. Hüseyin Demirci, Ihsan Taskin, Mustafa Çoban, and Adnan Baysal. Improved Meet-in-the-Middle Attacks on AES. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *LNCS*, pages 144–156. Springer, 2009.
13. W. Diffie and M. E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.
14. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-round AES. In Masayuki ABE, editor, *ASIACRYPT*, volume 6477 of *LNCS*, Singapore, December 2010. Springer. To appear, available `http://eprint.iacr.org/2010/322.pdf`.
15. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *LNCS*, pages 86–100. Springer, 2007.
16. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Bruce Schneier, editor, *FSE*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000.
17. Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, Singapore, December 2010. Springer.
18. Deukjo Hong, Bonwook Koo, and Yu Sasaki. Improved Preimage Attack for 68-Step HAS-160. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *LNCS*, pages 332–348. Springer, 2009.
19. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *LNCS*, pages 46–59. Springer, 2006.
20. Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A Practical Attack on KeeLoq. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 1–18. Springer, 2008.
21. Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. Printcipher: A block cipher for ic-printing. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

22. Gregor Leander, Christof Paar, Axel Poschmann, and Kai Schramm. New Lightweight DES Variants. In Alex Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 196–210. Springer, 2007.
23. Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In JooSeok Song, Taekyoung Kwon, and Moti Yung, editors, *WISA*, volume 3786 of *LNCS*, pages 243–258. Springer, 2005.
24. Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *LNCS*, pages 253–271. Springer, 2008.
25. Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.
26. Gautham Sekar, Nicky Mouha, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. *Information Processing Letters*. Submitted.
27. Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Aggelos Kiayias, editor, *CT-RSA*, Lecture Notes in Computer Science. Springer, 2011. To appear, available `http://www.cosic.esat.kuleuven.be/publications/article-1505.pdf`.

Appendix A The key schedule of the KTANTAN family

rounds	$(k_{a,r}, k_{b,r})$									
1 - 10	63 , 31	31 , 63	31 , 63	15 , 47	14 , 14	60 , 76	40 , 40	49 , 17	35 , 67	54 , 22
11 - 20	45 , 77	58 , 26	37 , 69	74 , 10	69 , 69	74 , 10	53 , 21	43 , 43	71 , 7	63 , 79
21 - 30	30 , 62	45 , 45	11 , 11	54 , 70	28 , 60	41 , 41	3 , 19	38 , 70	60 , 28	25 , 73
31 - 40	34 , 34	5 , 21	26 , 74	20 , 52	9 , 41	2 , 18	20 , 68	24 , 56	1 , 33	2 , 2
41 - 50	52 , 68	24 , 56	17 , 49	3 , 35	6 , 6	76 , 76	72 , 8	49 , 17	19 , 51	23 , 55
51 - 60	15 , 63	14 , 46	12 , 28	24 , 72	16 , 48	1 , 49	2 , 34	4 , 20	40 , 72	48 , 16
61 - 70	17 , 65	18 , 50	5 , 53	10 , 58	4 , 36	8 , 8	64 , 64	64 , 0	65 , 1	51 , 19
71 - 80	23 , 55	47 , 47	15 , 15	78 , 78	76 , 12	73 , 9	67 , 3	55 , 23	47 , 47	63 , 31
81 - 90	47 , 79	62 , 30	29 , 77	26 , 58	5 , 37	10 , 26	36 , 68	56 , 24	33 , 65	50 , 18
91 - 100	21 , 69	42 , 42	5 , 5	58 , 74	20 , 52	25 , 57	3 , 51	6 , 38	12 , 12	56 , 72
101 - 110	16 , 48	33 , 33	3 , 3	70 , 70	60 , 28	41 , 41	67 , 3	71 , 71	78 , 14	77 , 13
111 - 120	59 , 27	39 , 39	79 , 15	79 , 79	62 , 30	45 , 45	59 , 27	23 , 71	46 , 46	13 , 29
121 - 130	42 , 74	52 , 20	41 , 73	66 , 2	53 , 69	42 , 42	53 , 21	27 , 75	38 , 38	13 , 13
131 - 140	74 , 74	52 , 20	25 , 57	35 , 35	7 , 7	62 , 78	44 , 44	73 , 9	51 , 67	22 , 54
141 - 150	29 , 61	11 , 43	6 , 22	44 , 76	72 , 8	65 , 65	50 , 18	37 , 37	75 , 11	55 , 71
151 - 160	46 , 46	77 , 13	75 , 75	70 , 6	61 , 29	27 , 59	39 , 39	15 , 31	46 , 78	76 , 12
161 - 170	57 , 73	34 , 34	69 , 5	59 , 75	38 , 38	61 , 29	43 , 75	70 , 6	77 , 77	58 , 26
171 - 180	21 , 53	43 , 43	7 , 23	30 , 78	44 , 44	9 , 25	18 , 66	36 , 36	9 , 9	50 , 66
181 - 190	36 , 36	57 , 25	19 , 67	22 , 54	13 , 45	10 , 10	68 , 68	56 , 24	17 , 49	19 , 51
191 - 200	7 , 39	14 , 30	28 , 76	40 , 40	1 , 1	66 , 66	68 , 4	57 , 25	35 , 35	55 , 23
201 - 210	31 , 79	30 , 62	13 , 61	10 , 42	4 , 4	72 , 72	48 , 16	33 , 33	51 , 19	39 , 71
211 - 220	78 , 14	61 , 77	26 , 58	21 , 53	11 , 59	6 , 54	12 , 44	8 , 24	32 , 64	64 , 0
221 - 230	49 , 65	18 , 50	37 , 37	11 , 27	22 , 70	28 , 60	9 , 57	2 , 50	4 , 52	8 , 40
231 - 240	0 , 0	48 , 64	32 , 32	65 , 1	67 , 67	54 , 22	29 , 61	27 , 59	7 , 55	14 , 62
241 - 250	12 , 60	8 , 56	0 , 32	0 , 16	16 , 64	32 , 32	1 , 17	34 , 66	68 , 4	73 , 73
251 - 254	66 , 2	69 , 5	75 , 11	71 , 7						

Appendix B

Here we show the indirect partial matching of the optimal attack to KTANTAN32. When a state bit is affected by an active key bit, it is denoted + if linearly affected and 1 if nonlinearly affected. In the match, those bits with + can be matched with the indirect matching technique. It can be seen that the indirect matching technique gives 8 more bits match than the original 3 using direct partial matching.

```
forward part: 9 Neutral bits: 13 27 32 39 44 59 61 66 75
```

```
 forward R=110: k1=0, k2=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
 forward R=111: k1=1, k2=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 +
 forward R=112: k1=1, k2=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 + +
 forward R=113: k1=0, k2=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
 forward R=114: k1=0, k2=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 +
 forward R=115: k1=0, k2=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 + +


backward part: 6 Neutral bits: 3 20 41 47 63 74

backward R=147: k1=0, k2=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
backward R=146: k1=0, k2=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
backward R=145: k1=0, k2=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
backward R=144: k1=0, k2=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
backward R=143: k1=0, k2=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
backward R=142: k1=0, k2=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
backward R=141: k1=0, k2=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
backward R=140: k1=0, k2=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
backward R=139: k1=0, k2=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
backward R=138: k1=0, k2=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
backward R=137: k1=0, k2=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
backward R=136: k1=0, k2=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
backward R=135: k1=0, k2=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
backward R=134: k1=0, k2=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
backward R=133: k1=0, k2=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
backward R=132: k1=0, k2=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
backward R=131: k1=1, k2=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
backward R=130: k1=0, k2=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
backward R=129: k1=0, k2=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
backward R=128: k1=0, k2=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
backward R=127: k1=0, k2=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 0 0 0 0 0
backward R=126: k1=0, k2=0,  + 1 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
backward R=125: k1=0, k2=0,  0 + 1 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
backward R=124: k1=0, k2=0,  1 0 + 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 0 0
backward R=123: k1=1, k2=0,  1 1 0 + 1 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
backward R=122: k1=0, k2=1,  1 1 1 0 + 1 0 0 0 + 0 0 0 0  1 1 1 1 1 0 0 0 0 + + 0 0 0 0 0 0 0 0 0
backward R=121: k1=0, k2=1,  + 1 1 1 0 + 1 0 0 0 + 0 0 0  1 1 1 1 1 1 0 0 0 0 + + 0 0 0 0 0 0 0 0
backward R=120: k1=0, k2=0,  1 + 1 1 1 0 + 1 0 0 0 + 0 0  1 1 1 1 1 1 1 0 0 0 0 + + 0 0 0 0 0 0 0
backward R=119: k1=0, k2=0,  1 1 + 1 1 1 0 + 1 0 0 0 + 0  1 1 1 1 1 1 1 1 0 0 0 0 + + 0 0 0 0 0 0
backward R=118: k1=0, k2=0,  1 1 1 + 1 1 1 0 + 1 0 0 0 +  1 1 1 1 1 1 1 1 1 0 0 0 0 + + 0 0 0 0 0
backward R=117: k1=0, k2=0,  1 1 1 1 + 1 1 1 0 + 1 0 0 0  1 1 1 1 1 1 1 1 1 1 0 0 0 0 + + 0 0 0 0
backward R=116: k1=0, k2=0,  1 1 1 1 1 + 1 1 1 0 + 1 0 0  1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 + + 0 0 0


Bit map of 11 matched bits:  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
```