Match Box Meet-in-the-Middle Attack against KATAN

Thomas Fuhr and Brice Minaud

ANSSI, 51, boulevard de la Tour-Maubourg, 75700 Paris 07 SP, France
thomas.fuhr@ssi.gouv.fr,brice.minaud@gmail.com

Abstract. Recent years have seen considerable interest in lightweight cryptography. One particular consequence is a renewed study of meet-in-the-middle attacks, which aim to exploit the relatively simple key schedules often encountered in lightweight ciphers. In this paper we propose a new technique to extend the number of rounds covered by a meet-in-the-middle attack, called a match box. Furthermore, we demonstrate the use of this technique on the lightweight cipher KATAN, and obtain the best attack to date on all versions of KATAN. Specifically, we are able to attack 153 of the 254 rounds of KATAN32 with low data requirements, improving on the previous best attack on 115 rounds which requires the entire codebook.

Key words: Cryptanalysis, Meet-in-the-Middle, Biclique, Match Box, KATAN

1 Introduction

Over the past few years, ultra-lightweight embedded systems such as RFID tags and sensor nodes have become increasingly common. Many such devices require cryptography, typically for authentication purposes. However, traditional ciphers such as AES were not primarily designed for use in this context. Highly constrained devices impose a very small hardware footprint; on the other hand, they typically do not require a security level as high as that offered by AES.

To cater for this need, a number of lightweight ciphers have been developed, such as PRESENT [5], KATAN [7], LED [9], or Simon [2]. These ciphers aim to offer a trade-off between security and the constraints of embedded systems. This is often achieved by innovative designs that look to push the boundaries of traditional ciphers. The security of these new designs needs to be carefully assessed; in this process, new cryptanalytic techniques have emerged.

In particular, there has been a resurgence in the study of meet-in-the-middle attacks in the context of block ciphers [10, 6]. This type of attack requires a fairly simple key schedule, and is rarely applicable to traditional ciphers. However, many lightweight ciphers rely on simple round functions and key schedules, which are compensated by a high number of rounds. This makes them good targets for meet-in-the-middle attacks.

Our contribution.

In this paper, we propose a new way to extend meet-in-the-middle attacks, which we call a match box. This technique may be seen as a form of sieve-in-the-middle [8] or three-subset meet-int-the-middle attack [6], in that it extends the rounds covered in the middle section of the attack. It does so by relying on a large precomputed lookup table with a special structure. As such, it is also a form of time/memory trade-off.

We demonstrate this technique on the lightweight block cipher KATAN. As a result, we improve on previous results on all three versions of KATAN, both in terms of number of rounds as well as data requirements. Of independent interest is our construction of bicliques on KATAN, which takes full advantage of the linearity of the key schedule, and improves on previous attacks with negligible memory requirements

Related work.

Previous results on KATAN include a conditional differential analysis by Knellwolf, Meier and Naya-Plasencia [11] and a differential cryptanalysis of 115 rounds of KATAN32 by Albrecht and Leander [1]. In [10], Isobe and Shibutani describe meet-in-the-middle attacks on reduced versions of all three variants of KATAN. The attack that reaches the highest number of rounds on all three versions is a multidimensional meet-in-the-middle attack by Zhu and Gong [15]. However, this attack may be regarded as an optimized exhaustive search, as it involves performing a partial encryption under every possible value of the key.

Table 1 gives a summary of these results, including our own.

2 Meet-in-the-Middle attacks

2.1 Meet-in-the-Middle Framework

A meet-in-the-middle attack assumes that a few bits $\boldsymbol{v}$ of internal state may be computed from a plaintext by using a portion K_1 of the key; and that these same bits $\boldsymbol{v}$ may also be computed from the corresponding ciphertext with a portion K_2 of the key. The attack uses one plaintext/ciphertext pair as follows:

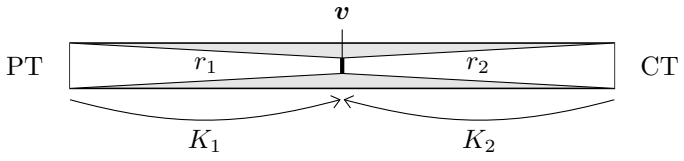

Fig. 1. Meet-in-the-middle attack.

– For each partial key $k_\cap \in K_\cap = K_1 \cap K_2$ [1]:

[1] This notation assumes, for the sake of simplicity, that the key schedule is linear (cf. §4.2). In general, the requirement is that once $K_\cap$ is guessed, the remaining information in K_1, and the remaining information in K_2 should be independent.

Table 1. Summary of results.

	Model	Data	Memory	Time	Rounds	Reference
KATAN32	CP	2^{22}	-	2^{22}	78	[11]
	KP	138	2^{75}	2^{77}	110	[10]
	CP	2^{32}	-	2^{79}	115	[1]
	KP	3	$2^{79.58}$	$2^{79.30}$	175	[15]
	KP	4	2^{5}	$2^{77.5}$	121	§4.3
	CP	2^{7}	2^{5}	$2^{77.5}$	131	§4.5
	CP	2^{5}	2^{76}	$2^{78.5}$	153	§4.7
KATAN48	CP	2^{34}	-	2^{34}	70	[11]
	KP	128	2^{78}	2^{78}	100	[10]
	KP	2	$2^{79.00}$	$2^{79.45}$	130	[15]
	KP	4	2^{5}	$2^{77.5}$	110	§4.3
	CP	2^{6}	2^{5}	$2^{77.5}$	114	§4.5
	CP	2^{5}	2^{76}	$2^{78.5}$	129	§4.7
KATAN64	CP	2^{35}	-	2^{35}	68	[11]
	KP	116	$2^{77.5}$	$2^{77.5}$	94	[10]
	KP	2	$2^{79.00}$	$2^{79.45}$	112	[15]
	KP	4	2^{5}	$2^{77.5}$	102	§4.3
	CP	2^{7}	2^{5}	$2^{77.5}$	107	§4.5
	CP	2^{5}	2^{74}	$2^{78.5}$	119	§4.7

- For each partial key $k_1 \in K_1$ extending $k_\cap$, $\boldsymbol{v}$ is computed. For each possible value of $\boldsymbol{v}$, the k_1's leading to that value are stored in a table.
- For each partial key $k_2 \in K_2$, $\boldsymbol{v}$ is computed. The k_1's leading to this same $\boldsymbol{v}$ are retrieved from the previous table. Each k_2 merged with each k_1 leading to the same $\boldsymbol{v}$ provides a candidate master key.

The actual encryption key is necessarily among candidate keys. Indeed, for the actual key, encryption from the plaintext and decryption from the ciphertext are mirrors of each other, and agree on the intermediate value $\boldsymbol{v}$. If we denote by $|\boldsymbol{v}|$ the size of $\boldsymbol{v}$, candidate keys form a proportion $2^{-|\boldsymbol{v}|}$ of the total key space.

In order to compute the actual encryption key, it remains to test candidate keys against enough plaintext/ciphertext pairs to ensure only one key remains. Each plaintext/ciphertext pair divides the number of candidates keys by $2^{|B|}$, where $|B|$ denotes the block size. Thus, in order to have only one key left, $\lceil |K|/|B| \rceil$ pairs are necessary on average, where $|K|$ denotes the key size.

In the end, the attack complexity in number of encryptions is:

$$2^{|K_\cap|} \cdot \left(2^{|K_1 - K_\cap|} \cdot \frac{r_1}{r} + 2^{|K_2 - K_\cap|} \cdot \frac{r_2}{r}\right) + \sum_{i=0}^{\lceil |K|/|B| \rceil - 1} 2^{|K| - |\boldsymbol{v}| - i|B|} \qquad (1)$$

where r_1 is the number of rounds in the encryption direction, r_2 is the number of rounds in the decryption direction, and $r = r_1 + r_2$.

Simultaneous matching. As we have seen, overall, meet-in-the-middle attacks proceed in two stages: a key filtering stage that produces key candidates, followed by a verification stage that tests the key candidates against a few plaintext/ciphertext pairs. This division in two stages is reflected in the complexity of the attack. The complexity of the first stage is determined mostly by the sizes of K_1 and K_2; the complexity of the second stage depends only on the size of $\boldsymbol{v}$ (for a fixed cipher).

Directly tweaking the size of $\boldsymbol{v}$ is one way to try and evenly spread the load between the two stages. However, increasing $\boldsymbol{v}$ will often disproportionately impact the sizes of K_1 and K_2. Simultaneous matching provides a very efficient alternate way of increasing the size of $\boldsymbol{v}$. The idea is to use n plaintext/ciphertext pairs instead of just one. For each guess of K_1 and K_2, we concatenate the $\boldsymbol{v}$'s produced by each pair in order to have a larger global $\boldsymbol{v}$, and use that for matching, as before.

In other words, what we are doing is perform a standard meet-in-the-middle attack, but on a cipher formed by n parallel applications of the basic cipher. This increases only linearly the complexity of the first stage, while exponentially decreasing the complexity of the second stage.

Indirect matching. With the newfound interest in meet-in-the-middle attack occasioned by lightweight ciphers, a number of techniques originally developed for the cryptanalysis of hash functions have been adapted to meet-in-the-middle attacks on block ciphers. A short survey of these techniques has already been presented in, for example, [14], and is out of the scope of this article. Still, we briefly mention one of these techniques, namely indirect matching, as we will use it later on KATAN. We also generalize this technique slightly.

In a regular meet-in-the-middle attack, some value $\boldsymbol{v}$ of the internal state is computed from the left as $e(k_1)$ and from the right as $d(k_2)$, where e and d are essentially a partial encryption and decryption. Keys are filtered by checking $e(k_1) = d(k_2)$. Now assume some key bit k in k_1 only has a linear impact on the value of $e(k_1)$, i.e. $e(k_1) = e'(k'_1) \oplus k$, where k'_1 is k_1 minus the knowledge of k. Then if knowledge of k is included in K_2, the equality in the middle $e(k_1) = d(k_2)$ may be rewritten as $e'(k'_1) = d(k_2) \oplus k = d'(k_2)$. In this way, guessing k is no longer necessary in the encryption direction, and the associated complexity decreases accordingly.

Here, we assumed that k is included in K_2, i.e. k is in $K_\cap$ since it is already in K_1. But we can get the same benefit even if k is in $K_1 - K_\cap$: the only real requirement is that it linearly impacts $e(k_1)$. To show this, the proof is a little more elaborate than in the previous case. Assume that k is in $K_1 - K_\cap$, and write $e(k_1) = e'(k'_1) \oplus k$ as before.

Up to now, k_1 together with k_2 was assumed to contain knowledge of the entire key. We guessed k_1 from the left, then k_2 from the right and matched compatible guesses by checking $e(k_1) = d(k_2)$. Instead, we are now going to guess k'_1 from the left and k_2 from the right, so the combination of the two does not encompass the entire key (k is missing). Furthermore, all guesses of k'_1 and

k_2 are compatible. However, for each pair of guesses, we set $k = e(k'_1) \oplus d(k_2)$, and the combination of k'_1, k_2 and k gives us one candidate master key.

Thus, the number of candidate master keys is unchanged. However, we need not guess k from the left, and the complexity of guessing k_1 is reduced accordingly. Thus the benefit is exactly the same as in the case where k belonged to $K_\cap$. Note that we remain compatible with simultaneous matching: if we use several plaintext/cipherext pairs, they all must agree on k, which yields the usual filter on the candidate master keys.

3 Match Box

We now introduce the match box technique. This technique fits within the general sieve-in-the-middle framework introduced in [8], which we recall here.

3.1 Sieve-in-the-middle

Let us still denote by $K_\cap$ the information on the key common to K_1 and K_2; furthermore, let K'_1 (resp. K'_2) be the proper part of K_1 (resp. K_2), i.e. the part not already in $K_\cap$. In a standard meet-in-the-middle attack, a few bits of internal state $\boldsymbol{l}$ are computed from the left by guessing $k_1 \in K_1$, then the same bits $\boldsymbol{r}$ are computed from the right by guessing $k_2 \in K_2$. Valid key candidates are determined by checking $\boldsymbol{l} = \boldsymbol{r}$.

However it would often be desirable to compute a few bits of information $\boldsymbol{l}$ from the left and $\boldsymbol{r}$ from the right, and discriminate keys by checking $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ for some general relation $\mathcal{R}$ expressing that $\boldsymbol{l}$ and $\boldsymbol{r}$ are compatible. It arises naturally if, say, $\boldsymbol{l}$ and $\boldsymbol{r}$ contain partial information about the internal state on either side of an S-box. In that case, $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ holds iff there exists an input/output pair of the S-box such that the input extends $\boldsymbol{l}$, and the output extends $\boldsymbol{r}$.

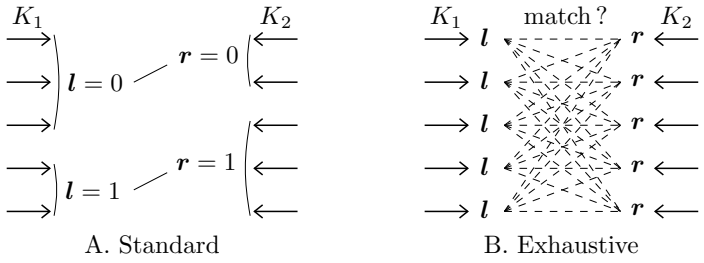

Fig. 2. Comparison of standard and exhaustive matching.

When applying this idea, the following problem arises. Once having guessed $k_\cap \in K_\cap$, the natural way to proceed would be to compute $\boldsymbol{l}$ and $\boldsymbol{r}$ for each $k'_1 \in K'_1$ and $k'_2 \in K'_2$ respectively, and exhaustively test $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ for every pair

(k'_1, k'_2). However, this would amount to a brute force search since $K_\cap \times K'_1 \times K'_2$ is in fact the entire key. It should be noted that there is no completely general solution to this problem, since $\mathcal{R}$ does need to be tested for every pair $(\boldsymbol{l}, \boldsymbol{r})$ yielded by every (k'_1, k'_2).

In the sieve-in-the-middle paper, this issue is solved by using merging algorithms originally introduced in [13]. These algorithms tend to assume, roughly, that the size of $\boldsymbol{l}$ is less than the size of K'_1 (divided by a sieving factor). We refer the reader to [8] for a complete explanation of merging techniques. What we propose is a different way of matching $\boldsymbol{l}$ and $\boldsymbol{r}$ while avoiding exhaustive search, which we call a match box.

3.2 Match Box

As we have mentioned in the previous section, the aim of the match box technique is to find compatible partial keys k_1 and k_2, such that the corresponding $\boldsymbol{l}$ and $\boldsymbol{r}$ satisfy a relation $\mathcal{R}$. The idea is to move the computation of $\mathcal{R}$ outside of the loop on $K_\cap$. In order to do this, we anticipate and precompute all possible matchings between $\boldsymbol{l}$ and $\boldsymbol{r}$. We start with an example.

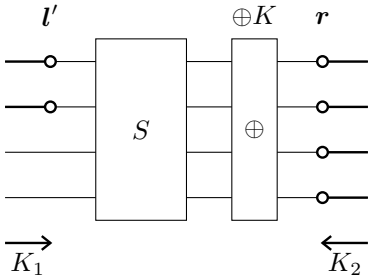

Fig. 3. A typical situation where a match box can apply.

Consider the situation depicted on Fig. 3. Here, $\boldsymbol{l}$ contains some partial information $\boldsymbol{l'}$ about the internal state entering an S-box. At the output of this S-box, some round key is added, and $\boldsymbol{r}$ contains the entire state after the key addition. Now assume that the round key may be decomposed as a sum of some $f_1(k'_1)$ depending on $k'_1 \in K'_1$, and some $f_2(k_2)$ depending on $k_2 \in K_2$. Note that this is automatically true if the key schedule is linear. Since K_2 is known when computing from the right, the component $f_2(k_2)$ may be directly added into $\boldsymbol{r}$.

So in this situation, $\boldsymbol{l} = (\boldsymbol{l'}, k'_1)$ and $\boldsymbol{r}$ are compatible iff $S^{-1}(\boldsymbol{r} \oplus f_1(k'_1))$ equals $\boldsymbol{l'}$ (wherever $\boldsymbol{l'}$ is defined). If $\boldsymbol{r}$ is larger than k'_2, since k'_1 is included in $\boldsymbol{l}$, $\boldsymbol{l}$ is also larger than k'_1, and a merging technique in the style of [8] cannot apply. However, a match box is possible.

In general, assume that $\boldsymbol{l} = (\boldsymbol{l'}, k'_1)$ contains the partial key $k'_1 \in K'_1$ plus some extra bits of information $\boldsymbol{l'}$, and $\boldsymbol{r}$ is as before. In order to anticipate all possible computations in the middle, we consider the function $f : k'_1 \mapsto \boldsymbol{l'}$ as a whole. For each value of this function, and each value of $\boldsymbol{r}$, we can precompute

a list of the k_1''s leading to $\boldsymbol{l'}$'s such that $\boldsymbol{l'}$, k_1', and $\boldsymbol{r}$ are compatible. Formally, let us denote by L' the set of values of $\boldsymbol{l'}$, and by R the set of values of $\boldsymbol{r}$. Then we precompute the following table, which we call a "match box":

$$\begin{aligned} M : {L'}^{K_1'} &\to {K_1'}^R \\ f &\mapsto \big(\boldsymbol{r} \mapsto \{k_1' : \mathcal{R}(\boldsymbol{l}, \boldsymbol{r})\}\big) \text{with } \boldsymbol{l} = (k_1',\, f(k_1')) \end{aligned}$$

This table takes as input a function $f : K_1' \to L'$, and produces as output the function that to each $\boldsymbol{r}$ associates all compatible k_1''s. Once this table has been precomputed, the attack proceeds as follows:

- For each $k_\cap \in K_\cap$:
  - For each $k_1' \in K_1'$, $\boldsymbol{l'}$ is computed. This yields a function $f : K_1' \to L'$, from which we obtain $M(f)$.
  - For each $k_2' \in K_2'$, $\boldsymbol{r}$ is computed. Candidate master keys are those corresponding to the pairs (k_1', k_2') for each k_1' in $M(f)(\boldsymbol{r})$.

The main limitation of this technique is the size of the table, which is approximated by:

$$2^{|l'|^{|K_1'|}+|r|+|K_1'|}$$

In particular, the size of K_1' (in terms of number of bits) must be exponentially small compared to the size of K. This is not surprising, since we are moving all computations of $\mathcal{R}$ outside of the loop on $K_\cap$: this constraint expresses the fact that there must be less possible situations in the middle than the size of the loop, otherwise we gain nothing.

3.3 Compressing $\mathcal{R}$

Looking more closely at the example in the previous section, the natural way to write $\mathcal{R}$ is in the form:

$$\begin{cases} l_1' &= f_1(k_1', \boldsymbol{r}) \\ l_2' &= f_2(k_1', \boldsymbol{r}) \\ \dots & \\ l_{|\boldsymbol{l'}|}' &= f_{|\boldsymbol{l'}|}(k_1', \boldsymbol{r}) \end{cases} \tag{2}$$

where the f_i's are boolean functions.

In this situation, each f_i is a boolean function of k_1', so it may be written as a polynomial in the bits of k_1'. As such, each f_i can be fully expressed by no more than $2^{|k_1'|}$ coefficients $(f_i^n)_{n<2^{|k_1'|}}$. This is beneficial as long as $|\boldsymbol{l'}| \cdot 2^{|k_1'|} < |\boldsymbol{r}|$, i.e. there are less f_i^n's than bits of $\boldsymbol{r}$. In this manner, $\boldsymbol{r}$ is effectively shortened to $|\boldsymbol{l'}| \cdot 2^{|k_1'|}$.

The only limit is the size and complexity necessary to build the table converting $\boldsymbol{r}$ into the f_i^n's. Note that in general, $\boldsymbol{r}$ is more or less a set of internal state bits, with potentially some partial keys added in; so computing $\boldsymbol{r}$ and the f_i's is akin to a partial encryption. In that case, for a given $\boldsymbol{r}$, the f_i^n's can be indirectly computed by evaluating the f_i's for all values of k_1'. In this way, for each $\boldsymbol{r}$ and each i, the value of the f_i^n's can be computed in at most $2^{|k_1'|}$ encryption equivalents.

4 Application to KATAN

KATAN is an ultra-lightweight block cipher presented by Christophe de Cannière, Orr Dunkelman and Miroslav Knežević at CHES 2009 [7]. Its design is inspired by the stream cipher Trivium, and relies on two nonlinear feedback registers. This is rather unique for a block cipher, and makes the cryptanalysis of KATAN especially interesting, since it indirectly evaluates the strength of this type of design.

In [7] the authors describe two families of block ciphers, KATAN and KTANTAN, which only differ in their key schedule. In KATAN, the key is stored in a register, while in KTANTAN, it is hardcoded into the circuit. The trade-off is that while the key cannot be modified, the circuit area is significantly reduced by avoiding the need for a register dedicated to the storage of the key. However, KTANTAN been broken [6, 14], mostly due to weaknesses in its key schedule. Hereafter we focus solely on KATAN.

4.1 Description of KATAN

KATAN is a family of three block ciphers with block sizes 32, 48, and 64 bits, denoted by KATAN32, KATAN48, and KATAN64 respectively. In all cases the key size is 80 bits, and the total number of rounds is 254. We begin by giving a brief description of KATAN32. KATAN48 and KATAN64 are very similar, as we shall see. We refer the reader to [7] for more details about the design of KATAN.

Key schedule. The master key is loaded into a 80-bit linear feedback register $(rk_0, \ldots, rk_{79})$, and new round keys are generated by the linear feedback relation:

$$rk_{i+80} = rk_i \oplus rk_{i+19} \oplus rk_{i+30} \oplus rk_{i+67}, \quad 0 \leq i \leq 428 \tag{3}$$

Round function. The 32-bit plaintext is loaded into two registers A and B of sizes 13 and 19 bits. The round function depicted on Fig. 4 is then applied 254 times, where c_n is a round constant defined by $(c_0, \ldots, c_7) = (1, \ldots, 1, 0)$ and $c_{i+8} = c_i \oplus c_{i+1} \oplus c_{i+3} \oplus c_{i+5}$.

Formally, KATAN32 encryption may be defined as follows. By a_n (resp. b_n), we denote the bit entering register A (resp. B) at round n. Hence, after round n, the content of register A is $(a_{n-12}, \ldots, a_n)$, and the content of register B is $(b_{n-18}, \ldots, b_n)$. By convention, the plaintext is $(a_{-13}, \ldots, a_{-1}, b_{-19}, \ldots, b_{-1})$. Then encryption is recursively defined by:

$$\begin{cases} a_n = b_{n-19} \oplus b_{n-8} \oplus b_{n-11} \cdot b_{n-13} \oplus b_{n-4} \cdot b_{n-9} \oplus rk_{2n+1} \\ b_n = a_{n-13} \oplus a_{n-8} \oplus c_n \cdot a_{n-4} \oplus a_{n-6} \cdot a_{n-9} \oplus rk_{2n} \end{cases} \tag{4}$$

and the ciphertext is $(a_{241}, \ldots, a_{253}, b_{235}, \ldots, b_{253})$.

KATAN48. KATAN48 uses two registers of sizes 19 and 29 bits. The registers are updated *twice* per round by the following feedback relation, using the same round keys:

$$\begin{cases} a_n = b_{n-29} \oplus b_{n-20} \oplus b_{n-14} \cdot b_{n-22} \oplus b_{n-7} \cdot b_{n-16} \oplus rk_{2 \cdot \lfloor \frac{n}{2} \rfloor + 1} \\ b_n = a_{n-19} \oplus a_{n-13} \oplus c_n \cdot a_{n-7} \oplus a_{n-8} \cdot a_{n-16} \oplus rk_{2 \cdot \lfloor \frac{n}{2} \rfloor} \end{cases}$$

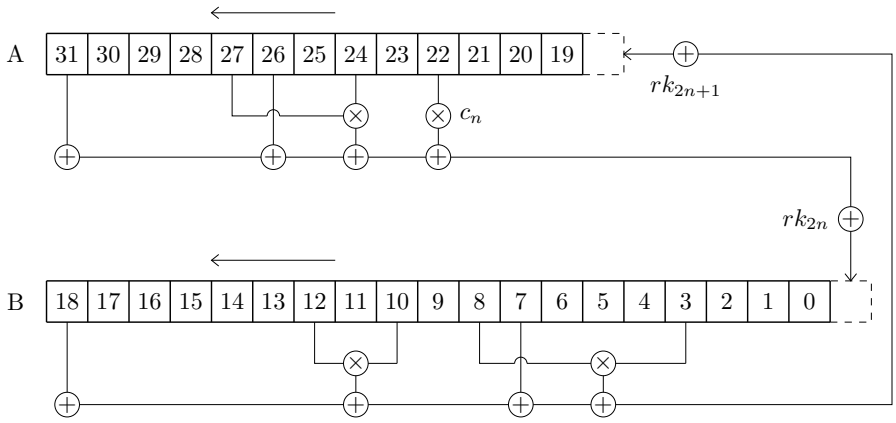

Fig. 4. Round function of KATAN32.

After 254 rounds, the ciphertext is $(a_{489}, \ldots, a_{507}, b_{479}, \ldots, b_{507})$.

KATAN64. KATAN64 uses two registers of sizes 25 and 39 bits. The registers are updated *three times* per round by the following feedback relation, using the same round keys:

$$\begin{cases} a_n = b_{n-39} \oplus b_{n-26} \oplus b_{n-22} \cdot b_{n-34} \oplus b_{n-10} \cdot b_{n-15} \oplus rk_{2 \cdot \lfloor \frac{n}{3} \rfloor + 1} \\ b_n = a_{n-25} \oplus a_{n-16} \oplus c_n \cdot a_{n-10} \oplus a_{n-12} \cdot a_{n-21} \oplus rk_{2 \cdot \lfloor \frac{n}{3} \rfloor} \end{cases}$$

After 254 rounds, the ciphertext is $(a_{737}, \ldots, a_{761}, b_{723}, \ldots, b_{761})$.

4.2 Linear Key Partition

We now introduce a few notions that will prove useful to mount a meet-in-the-middle attack against KATAN. Let RK_1 (resp. RK_2) denote the set of round keys necessary to compute some fixed bits of internal state at an intermediate round from the left (resp. from the right). The first step of a meet-in-the-middle attack is to guess the bits of information on the master key common to RK_1 and RK_2 (see §2.1). Hence it is necessary to define an intersection of RK_1 and RK_2 in terms of bits of information on the master key.

In general, this intersection may be impossible to define. In [10], a generic solution is proposed: all round keys are regarded as independent, i.e. the master key is redefined as the union of all round keys. This yields good results on various lightweight ciphers, including KATAN. However, it has a significant impact on the attack complexity. This can be avoided when the key schedule is linear: indeed, in that case, the intersection of RK_1 and RK_2 can be cleanly defined, as we now show for KATAN.

Let us regard a master key of KATAN as a vector in $E = (\mathbb{Z}/2\mathbb{Z})^{80}$. The value of the master key corresponds to the coordinates of this vector along the canonical basis. Each round key is a linear combination of bits of the master key;

that is, it is the image of the master key through some map $(x_i) \mapsto \sum \lambda_i x_i$, i.e. a linear functional on E. Let us denote by $\mathcal{L}(E)$ the space of linear functionals on E.

From this standpoint, the information carried by RK_1 (resp. RK_2) is the value of the master key on the subspace E_{K_1} (resp. E_{K_2}) of $\mathcal{L}(E)$ generated by the round keys of RK_1 (resp. RK_2). Let $E_{K_\cap} = E_{K_1} \cap E_{K_2}$. Then the bits of information on the master key common to RK_1 and RK_2 are exactly the value of the key on the functionals of $E_{K_\cap}$.

Let us choose an arbitrary basis $B_\cap$ of $E_{K_\cap}$, and extend it to a basis B_1 of E_{K_1}, and B_2 of E_{K_2}. Then in concrete terms a partial key in $K_\cap$ is a mapping $B_\cap \to \{0,1\}$; likewise, K_1 and K_2 are regarded as the set of mappings $B_1 \to \{0,1\}$ and $B_2 \to \{0,1\}$ respectively. We are now able to apply the meet-in-the-middle attack framework exactly as it was presented in section 2.1.

In the remainder, it will always be assumed that $B = B_1 \cup B_2$ is a basis for the whole space $\mathcal{L}(E)$. In particular, knowledge of the value of a key on B amounts to knowing the entire key; it will be convenient at times to identify the key space with $\{0,1\}^B$, which we will denote by K, by analogy with K_1 and K_2.

4.3 Key dependencies

A first step towards building a meet-in-the-middle attack is to choose a value $\boldsymbol{v}$ extracted from an internal state at an intermediate round to serve as a meeting point. In order to make this choice, it is necessary to evaluate which key bits are necessary to compute $\boldsymbol{v}$ from the plaintext, and from the ciphertext (presumably for some reduced version of the cipher). We have carried out this computation using an algorithm similar to Algorithm 1 in [10].

The principle of such an algorithm is that once some round key enters the state, the impacted bit is marked as depending on that key. Then this dependency is propagated along the cipher each time this internal state bit affects other internal state bits. In our case, because we will use indirect matching, we keep track separately of key bits whose impact is linear, and those whose impact is nonlinear.

By nature, such an analysis follows a worst case scenario, i.e. it assumes any key bit than could possibly affect an internal state bit, does. In reality, fortuitous simplifications may occur. However, in the case of KATAN, our algorithm was precise enough that experimental tests observed the same dependencies between internal state bits and key bits. Table 2 shows our results for KATAN32.

Basic meet-in-the-middle attack against KATAN.

With what we have so far, we can mount a first meet-in-the-middle attack against KATAN. While this is not the best attack we will propose, it is still worth mentioning because it has a simple description, requires only known plaintexts and minimal data requirements, and improves on previously published attacks.

For KATAN32, if we aim at a complexity around 2^{77}, we can attack $60+61 = 121$ rounds (cf. Table 2). The meeting point is b_{50} (which is indeed at position 9 of register B after 60 rounds). The dimensions of K_1, K_2, $K_\cap$ are 75, 75, 70 respectively (after ignoring linear contributions thanks to indirect matching).

Table 2. Key dependency of the bit at position 9 (middle) of register B.

KATAN32 Encryption (starting from round 0):

Number of rounds		58	59	60	61	62	63	64	65	66
Dimension of	nonlinear	70	71	75	77	78	80	78	79	80
key space	linear	2	2	2	2	2	0	1	1	0

KATAN32 Decryption (starting from round 254):

Number of rounds		57	58	59	60	61	62	63	64	65
Dimension of	nonlinear	68	70	72	73	75	76	78	79	80
key space	linear	4	3	3	2	1	1	0	0	0

We use 4 plaintext/ciphertext pairs for simultaneous matching to ensure that the key verification stage is in 2^{76}. Using (1), and taking into account that we use 4 plaintext/ciphertext pairs, the overall complexity is:

$$4 \cdot \left(2^{75} \cdot \frac{60}{121} + 2^{75} \cdot \frac{61}{121} \right) + \sum_{i=0}^{2} 2^{76-32i} \approx 2^{77.5}$$

In a similar way, we can attack $56+54 = 110$ rounds of KATAN48, and $51+51 = 102$ rounds of KATAN64, both of them with 4 plaintext/ciphertext pairs and complexity $2^{77.5}$.

4.4 Bicliques

The number of rounds covered by a meet-in-the-middle attack may be extended by a biclique. This technique was also originally developed for the cryptanalysis of hash functions [4], and first applied to block ciphers in [3] to produce an accelerated key search against AES. Such a search requires all possible keys to be tried, but each try costs significantly less than a full encryption.

However, bicliques may also be used in the context of a traditional attack, where not all keys are tried. This is the model known as "long bicliques" in [3], and corresponds to [4] for hash functions. We will use this approach against KATAN, and so we recall it here briefly.

Definition 1. *A* biclique *is a triple* $((A_i)_{i \le n}, (B_i)_{i \le n}, (K_{i,j})_{i,j \le n})$*, where the* A_i*'s are internal states at some round* a*, the* B_i*'s are internal states at some round* b*, and the* $(K_{i,j})$*'s are keys satisfying the following property:*

$$\forall i, j \le n, \quad \mathrm{Enc}_{K_{i,j}}^{a \to b}(A_i) = B_j$$

where $\mathrm{Enc}_{K_{i,j}}^{a \to b}$ *denotes encryption from round* a *to round* b *with key* $K_{i,j}$*.*

For simplicity, assume $a = 0$, and we have a biclique covering rounds a to b as in the above definition. In order to construct an attack up to round r, the remaining rounds from b to r must be covered by a meet-in-the-middle

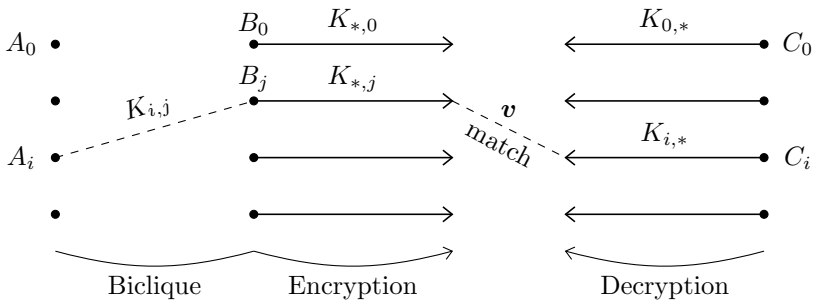

Fig. 5. A meet-in-the-middle attack and compatible biclique.

attack. Furthermore, the biclique and the meet-in-the-middle segments must be compatible in the following sense. Let C_i be the ciphertext corresponding to A_i after r encryption rounds, et $\boldsymbol{v}$ be the internal value used as a meeting point for the meet-in-the-middle attack. Let $K_{i,*}$ denote the partial information on the key expressing the fact that it is one of the $K_{i,j}$'s, for fixed i and variable j. Let $K_{*,j}$ be defined in the same way.

Then the biclique and the meet-in-the-middle segments of the attack are *compatible* iff the middle value $\boldsymbol{v}$ can be computed starting from B_j with only knowledge of $K_{*,j}$, and from C_i with only knowledge of $K_{i,*}$. The situation is illustrated on Fig. 5. This requirement is quite restrictive. However, it becomes easier to enforce if the key schedule is linear, as we shall see with KATAN.

Attack process.

- For each partial key $k_\cap \in K_\cap = \bigcup K_{i,*} \cap K_{*,j}$:
  - For each $j \leq n$, $\boldsymbol{v}$ is computed starting from B_j using $K_{*,j}$. For each possible value of $\boldsymbol{v}$, the j's leading to that value are stored in a table.
  - For each $i \leq n$, $\boldsymbol{v}$ is computed starting from C_i using $K_{i,*}$. The j's leading to this same $\boldsymbol{v}$ are retrieved from the previous table. For each pair (i, j) leading to the same $\boldsymbol{v}$, $K_{i,j}$ is a candidate master key.

If the actual encryption key is among the $K_{i,j}$'s, then it is necessarily a candidate. Indeed, encryption by $K_{i,j}$ will follow the path depicted on Fig. 5. As with a standard meet-in-the-middle attack, it remains to test candidate keys on a few additional plaintext/ciphertext pairs to single out the right key. This step is unchanged. Finally, to ensure that the actual key is among the $K_{i,j}$'s, the key space must be covered by bicliques, and the previous attack is repeated for each biclique.

Construction of a biclique varies depending on the cipher, but in general the construction cost is negligible with respect to the global complexity. Note that it is implicitly assumed that the construction of a biclique on a set of keys $K_{i,j}$ does not imply that each key be computed, i.e. there is a structure. The overall complexity is then the same as that of the meet-in-the middle segment if it were simply applied to a fixed plaintext/ciphertext pair.

4.5 Bicliques on KATAN

We have presented bicliques in the previous section. It remains to show how to construct bicliques on KATAN. Once again, it all comes down to the linearity of the key schedule, and the weak non-linearity of the cipher reduced to a few rounds. In fact, these two properties make it possible to adjoin a biclique to any pre-existing, arbitrary meet-in-the-middle attack, in a compatible manner. Furthermore, a single biclique will suffice to cover the entire key space.

Assume we have a pre-existing meet-in-the-middle attack, with the notation of the previous sections. Recall that $K_\cap$ (resp. K_1, K_2) are regarded as maps $B_\cap \to \{0,1\}$ (resp. $B_1 \to \{0,1\}$, $B_2 \to \{0,1\}$). Let us denote by K_1' (resp. K_2') the proper part of K_1 (resp. K_2) with respect to $K_\cap$, i.e. its restriction to $B_1 - B_\cap$ (resp. $B_1 - B_\cap$).

Let us denote by $\mathrm{Enc}_{a\to b}^k M$ and $\mathrm{Dec}_{b\to a}^k M$ the encryption and decryption of a message M between rounds a and b with key k. We extend this notation to the case where k is a partial key (i.e. an element of K_1, K_1', K_2, K_2', or $K_\cap$) by completing the key by 0 on the rest of B. In addition, for $k \in K$, let us write $k_1 \in K_1$ for its restriction to B_1, and define in a similar way k_1', k_2, and k_2'. Finally, let $k(i)$ denote the value of the i-th round key generated by k ; again, if k is only partially defined, it is completed by 0 on the rest of the basis B.

Definition 2. *Let :*

$$\begin{aligned}
&\mathrm{Bic}_n(K_1, K_2) = ((A_{k_2} : k_2 \in K_2), (B_{k_1} : k_1 \in K_1), K_2' \oplus K_1) \\
&\text{With :} \quad A_{k_2} = \mathrm{Dec}_{n\to 0}^{k_2'}(0) \\
&\qquad\quad\ \ B_{k_1} = \mathrm{Enc}_{0\to n}^{k_1}(0)
\end{aligned}$$

where 0 *is the null block. Observe that* A_{k_2} *is decrypted by the projection* k_2'*.*

Proposition 1. *For each version of* KATAN*, there exists* $n > 0$ *such that for any* K_1*,* K_2*,* $\mathrm{Bic}_n(K_1, K_2)$ *is a biclique. That is, for* $k \in K$*, with* k_1 *and* k_2 *its projections on* K_1 *and* K_2*:*

$$\mathrm{Enc}_{0\to n}^k(A_{k_2}) = B_{k_1} \tag{5}$$

Proof. The proof is essentially the same for all three versions of KATAN. For the sake of simplicity, we only present the proof for KATAN32. It will be convenient to designate bits in each register by their position, in the order depicted on Fig. 4.

We claim that the proposition holds for $n = 10$. The core of the proof lies in the following property:

$$\forall i \le 10, \quad \mathrm{Enc}_{0\to i}^k(A_{k_2}) = \mathrm{Enc}_{0\to i}^{k_1}(0) \oplus \mathrm{Dec}_{10\to i}^{k_2'}(0) \tag{6}$$

For $i = 10$, this equation becomes:

$$\mathrm{Enc}_{0\to 10}^k(A_{k_2}) = \mathrm{Enc}_{0\to 10}^{k_1}(0) = B_{k_1}$$

which is precisely what we want to prove. So (6) implies the proposition. We are going to prove (6) by recursion on $0 \leq i \leq 10$.

For $i = 0$, (6) yields the definition of A_{k_2}, so it holds. Assume that it holds for some round $i < 10$. When we step forward one encryption round, since we are dealing with shift registers, the equality remains true everywhere, except possibly on the two new bits entering the registers (at positions 0 and 19 on Fig. 4). Let us show for instance that the equality remains true for the bit entering register B (position 0). Let us denote by f the feedback function from register A into register B.

Then for the bit entering register B, (6) at round $i + 1$ means:

$$f\big(\mathrm{Enc}^{k}_{0\to i}(A_{k_2})\big) \oplus k(i) = f\big(\mathrm{Enc}^{k_1}_{0\to i}(0)\big) \oplus k_1(i) \oplus f\big(\mathrm{Dec}^{k'_2}_{10\to i}(0)\big) \oplus k'_2(i) \quad (7)$$

where $k(i)$ denotes the value of the i-th round key generated by key k. Since $k = k_1 \oplus k'_2$, using the recursion hypothesis, we get :

$$f\big(\mathrm{Enc}^{k_1}_{0\to i}(0) \oplus \mathrm{Dec}^{k'_2}_{10\to i}(0)\big) = f\big(\mathrm{Enc}^{k_1}_{0\to i}(0)\big) \oplus f\big(\mathrm{Dec}^{k'_2}_{10\to i}(0)\big) \quad (8)$$

Since f is nonlinear, this is not automatically true. However, the only nonlinear interaction in f is a multiplication of bits 24 and 27. Now observe that $\mathrm{Enc}^{k_1}_{0\to i}(0)$ is 0 on bits $19 + i, \ldots, 31$, and $\mathrm{Dec}^{k'_2}_{10\to i}(0)$ is 0 on bits $19, \ldots, 21 + i$. Hence for $n \leq 5$, $\mathrm{Enc}^{k_1}_{0\to i}(0)$ is 0 on bits 24 and 27, and for $n \geq 6$, $\mathrm{Dec}^{k'_2}_{10\to i}(0)$ is 0 on bits 24 et 27.

Assume for instance we are in the first case. Then in (8), bits 24 and 27 are equal for $\mathrm{Enc}^{k_1}_{0\to i}(0) \oplus \mathrm{Dec}^{k'_2}_{10\to i}(0)$ and $\mathrm{Dec}^{k'_2}_{10\to i}(0)$, and null for the last term, thus the only nonlinear component of f has the same contribution on each side of the equation. On the rest f is linear, so we are done.□

In essence, there is only one multiplication on register A of KATAN32, between bits 24 and 27. By restricting ourselves to $13 - (27 - 24) = 10$ rounds, where 13 is the length of register A, we ensure that there is no nonlinear interaction between the bits of B_{k_1} dependent on k_1, and the bits of $A_{k'_2}$ dependent on k'_2. With register B the same computation yields $19 - \max(12 - 10, 8 - 3) = 14$ rounds. That is why we can build a biclique of length 10 on KATAN32. The same reasoning shows that we can build bicliques of length 5 on KATAN48, and 5 again on KATAN64.

Building several bicliques.

Later on, we will want to use simultaneous matching (cf. §2.1). For this purpose, we need several distinct bicliques, and the previous proposition only gives us one. Fortunately, it is possible to build new distinct bicliques on the same model as that of Proposition 1, by adding parameters to Definition 2. There are several ways to proceed. In particular, it is possible to either modify the bits of A_{k_2} that do not actually depend on k_2, or those that do. We only describe the second option, as it is enough for our purpose.

Pick any arbitrary key k_P as parameter. In fact, only the first ten pairs of round keys derived from k_P will have an impact, so we have 20 degrees of

freedom. Then define A_{k_2} and B_{k_1} by:

$$A_{k_2} = \mathrm{Dec}_{n\to 0}^{k_2' \oplus k_P}(0)$$
$$B_{k_1} = \mathrm{Enc}_{0\to n}^{k_1 \oplus k_P}(0)$$

and replace the recursion equation (6) in the proof by:

$$\forall i \leq 10, \quad \mathrm{Enc}_{0\to i}^{k}(A_{k_2}) = \mathrm{Enc}_{0\to i}^{k_1 \oplus k_P}(0) \oplus \mathrm{Dec}_{10\to i}^{k_2' \oplus k_P}(0)$$

The proof is exactly the same, except in (7), the contribution of k_P on the right-hand side cancels itself out.

Biclique attack against KATAN.

In §4.3, we attacked 121 rounds of KATAN32. If we use four bicliques instead of four plaintext/ciphertext pairs, we gain an additional 10 rounds, as explained in the previous section. Meanwhile, the core of the attack remains the same, except we meet on b_{60} starting from round 10, instead of b_{50} starting from round 0. In particular, the complexity is unchanged. However we now require chosen plaintexts. Because the dimension of K_2' is 5, each biclique requires 2^5 chosen plaintexts, so the data requirements increase to $4 \cdot 2^5 = 2^7$. The attack covers 131 rounds with complexity $2^{77.5}$. In the same way, we can extend the previous attacks on KATAN48 and KATAN64 respectively to 114 rounds with 2^6 CP, and 107 rounds with 2^7 CP, both with complexity $2^{77.5}$.

4.6 Match Box

We now explain how the match box technique applies to KATAN32. Variants for KATAN48 and KATAN64 will be very similar. Assume we are meeting in the middle on b_{62} (this will be the case in the final attack). The idea is that we are going to isolate round keys whose impact on the value of b_{62} when computing from the right can be evaluated with knowledge of only a few bits of information. We do not consider round keys whose impact is only linear however, since those can be ignored thanks to indirect matching.

When decrypting from the right, the value of b_{62} is (cf. (4)):

$$\begin{aligned} b_{62} &= a_{81} \oplus b_{73} \oplus b_{68} \cdot b_{70} \oplus b_{72} \cdot b_{77} \oplus rk_{163} \\ &= x_0 \oplus b_{68} \cdot b_{70} \qquad \text{with } x_0 = a_{81} \oplus b_{73} \oplus b_{72} \cdot b_{77} \oplus rk_{163} \end{aligned} \tag{9}$$

Let us further decompose b_{68} and b_{70} in the above formula as:

$$b_{68} = x_1 \oplus rk_{175}, \quad (x_1 = a_{87} \oplus b_{89} \oplus b_{76} \cdot b_{74} \oplus b_{83} \cdot b_{78})$$
$$b_{70} = x_2 \oplus rk_{179}, \quad (x_2 = a_{89} \oplus b_{91} \oplus b_{78} \cdot b_{76} \oplus b_{85} \cdot b_{80})$$

Since $K_2 \oplus K_1' = K$, each round key rk_n may be written as $rk_n = rk_n^2 \oplus rk_n^{1'}$, with $rk_n^2 \in K_2$ and $rk_n^{1'} \in K_1'$. We define $\boldsymbol{r}$ as:

$$r_0 = x_0$$
$$r_1 = x_1 \oplus rk_{175}^2$$
$$r_2 = x_2 \oplus rk_{179}^2$$

Putting everything together, $\boldsymbol{l}$ contains $a_0 = b_{62}$ computed from the left, as well as $k'_1 \in K'_1$; and $\boldsymbol{r}$ is (r_0, r_1, r_2). The matching relation $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ witnessing the fact that computations from either side agree on b_{62} is:

$$\mathcal{R}(\boldsymbol{l}, \boldsymbol{r}) : \quad l_0 = r_0 \oplus (r_1 \oplus k'_1(rk^{1'}_{175})) \cdot (r_2 \oplus k'_1(rk^{1'}_{179}))$$

Note that this is merely a rewriting of (9), where the contribution of round keys 175 and 179 has been isolated, and then split in order to extract their K'_1 component.

What we have gained in this example is that round keys 175 and 179 no longer need to be known when computing from the right. This decreases the dimension of K_2 by two, and thus spares a factor 2^2 when guessing its value. Moreover this gain can be spent in order to extend the attack to one more round. Indeed, we can now append one extra round at the end of the (reduced) cipher, and simply add the two bits of round key for that round into K_2. This increases the dimension of K_2 by two, back to its original value. Then the attack proceeds as before. In short, every time we are able to decrease the dimension of K_2 by two by needing less round keys in order to compute r_{62}, we can re-increase it in order to extend the attack to one more round.

Thus, to extend the attack further, we need only isolate the contribution of more round keys in (9), as we have done we round keys 175 and 179. The limit is the size of $\boldsymbol{r}$, which impacts the size of the match box table. For example, the next step would be to expand $b_{72} \cdot b_{77}$ in x_0, in the same manner we have previously expanded $b_{68} \cdot b_{70}$ in b_{62}. That is, we develop the expression of these two bits according to (4), making round keys rk_{183} and rk_{193} appear:

$$\begin{aligned} b_{72} &= x_3 \oplus rk_{183}, \quad (x_3 = a_{91} \oplus b_{93} \oplus b_{78} \cdot b_{80} \oplus b_{82} \cdot b_{87}) \\ b_{77} &= x_4 \oplus rk_{193}, \quad (x_4 = a_{96} \oplus b_{98} \oplus b_{83} \cdot b_{85} \oplus b_{87} \cdot b_{92}) \end{aligned}$$

We now define a new 5-bit $\boldsymbol{r}$ by:

$$\begin{aligned} r'_0 &= x_0 \oplus b_{72} \cdot b_{77} = a_{81} \oplus b_{73} \oplus rk_{163} \\ r_1 &= x_1 \oplus rk^2_{175} \\ r_2 &= x_2 \oplus rk^2_{179} \\ r_3 &= x_3 \oplus rk^2_{183} \\ r_4 &= x_4 \oplus rk^2_{193} \end{aligned}$$

And the relation $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ becomes :

$$l_0 = r'_0 \oplus \left(r_1 \oplus k'_1(rk^{1'}_{175})\right) \cdot \left(r_2 \oplus k'_1(rk^{1'}_{179})\right) \oplus \left(r_3 \oplus k'_1(rk^{1'}_{183})\right) \cdot \left(r_4 \oplus k'_1(rk^{1'}_{193})\right)$$

It so happens that rk_{175} and rk_{179} (as well as rk_{183} and rk_{193}) only appear in one place in the development of b_{62}. Due to the diffusion of the cipher, later round keys will appear in several places in the expansion of b_{62}. As a result the "cost" for each new round key in terms of the increase in the size of $\boldsymbol{r}$ will grow. In order to choose which round keys to isolate, we have implemented a greedy

Table 3. Sizes of $\boldsymbol{r}$ sufficient to spare a certain number of round keys.

KATAN32 (starting from r_{62}):

$\lvert\boldsymbol{r}\rvert$	3	5	7	9	11	15	17	23	27	31	39	43	53	61	65	71	73
Round keys	2	5	7	9	10	12	13	16	18	20	23	24	28	30	32	34	35

KATAN48 (starting from r_{109}):

$\lvert\boldsymbol{r}\rvert$	5	9	17	21	25	27	35	43	55	63	73
Round keys	3	5	7	8	10	12	14	16	18	20	22

KATAN64 (starting from r_{153}):

$\lvert\boldsymbol{r}\rvert$	7	11	15	17	25	33	45	65	71
Round keys	3	5	7	8	10	12	14	17	18

algorithm that essentially adds the cheapest round key (in terms of the growth of $\boldsymbol{r}$) at each step. Results are shown on Table 3.

Note that this table only indicates the number of round keys spared. One can expect that every two round keys spared gains one round, but this is dependent on the two round keys being linearly independent of the rest of K_2. This in turn depends on which round keys are in K_2, i.e. which rounds are covered by K_2.

4.7 Final attack

In this section, we describe the final version of the attack we propose against KATAN32, combining all components from the previous sections. We aim at having K_1 and K_2 both of dimension 77. Starting from round 10 after the biclique, this allows us to cover 62 rounds in the forward direction. This corresponds to meeting on b_{62} (i.e. position 9 of register B after 72 rounds). The number of rounds covered in the backwards direction will depend on the match box. We will ensure that in the end, the dimension of K_1' is 3.

Compression table (cf. §3.3). We have $|k_1'| = 3$, so $2^{|k_1'|} = 8$, which makes it worthwhile to use the compression technique. We want to build a compression table C converting a $\boldsymbol{r}$ of size greater than 8 into 8 f^n's. Note that we meet on a single bit, so there is only one line in (2), which is why we talk about f^n's and not f_i^n's. On the other hand, we will use simultaneous matching on three bicliques, but always on the same bit, so the conversion from $\boldsymbol{r}$ into the f^n's is the same for each pair: we only need one compression table.

As observed in §3.3, for each $\boldsymbol{r}$, the f^n's can be computed with $2^{|k_1'|}$ partial encryptions. Hence for KATAN32 we can choose $\boldsymbol{r} = 73$ (see Table 3), yielding a table of size 2^{76} in complexity 2^{76}. This spares 35 round keys in the decryption direction. In the end, we can begin the backwards computation from round 153.

Match box (cf. §4.6). We perform simultaneous matching on b_{62} for 3 distinct bicliques; $\boldsymbol{l'}$ contains the value of b_{62} computed from the left for each biclique, so $|\boldsymbol{l}| = 3$. Meanwhile $\boldsymbol{r}$ contains the 8 bits f^n computed from the right, again for each biclique, hence $|\boldsymbol{r}| = 8 \times 3 = 24$. This yields a match box table of size $2^{3^3+24+3} = 2^{54}$ in less than 2^{54} encryptions. Note that both the compression

table and the match box table are absolute precomputations, in the sense that they do not depend on the actual plaintext/ciphertext pairs and need only be built once.

In the end, we attack 153 rounds: the first 10 are covered by the bicliques; the next 71 are the forward part of the meet-in-the-middle attack; the next 19 are covered by the match box; and the final 53 are the backwards part of the meet-in-the-middle attack. See the Appendix for the list of round keys involved in K_1 and K_2 and the list of round keys spared by using the match box.

Attack process.

- Precompute the compression table C.
- Precompute the match box M.
- For each partial key $k_\cap \in K_\cap$:
  - For each partial key k'_1, knowing $k_1 = k_\cap \oplus k'_1$, compute b_{62} from the left for each biclique, and denote their concatenation by $\boldsymbol{l'}$.
    This yields a function $F : k'_1 \to \boldsymbol{l'}$. Retrieve $M(F)$.
  - For each partial key k'_2:
    - For each biclique, knowing $k_2 = k_\cap \oplus k'_2$, compute the 31-bit $\boldsymbol{r}$ from the right for that biclique. Convert it into 8 bits f^n through C.
    - Having done this for all 3 bicliques, the concatenation of the f^n's makes up the 24-bit $\boldsymbol{r}$ entry of the match box. Match k'_2 with the k'_1's in $M(F)(\boldsymbol{r})$ to form candidate master keys.
- Test candidates master keys on 3 plaintext/ciphertext pairs as in a standard meet-in-the-middle attack. This should be done on the fly.

The overall attack complexity is:

$$2^{74} + 2^{54} + 3 \cdot \left(2^{77} \cdot \frac{62}{153} + 2^{77} \cdot \frac{81}{153}\right) + \sum_{i=0}^{2} 2^{77-32i} \approx 2^{78.5}$$

For KATAN48, aiming at the same complexity, we can cover $57 + 56 = 113$ rounds with the meet-in-the-middle section, 5 rounds with the bicliques, and an additional 11 rounds with the match box, for a total of 129 rounds. For KATAN64, the bicliques cover 5 rounds as well, the match box 9, and the meet-in-the-middle portion of the attack reaches $52 + 53 = 105$ rounds, for a total of 119 rounds. The complexity and data requirements are shown on Table 1.

5 Conclusion

In this paper, we presented a new technique to extend meet-in-the-middle attacks. This technique makes it possible to extend the middle portion of the attack with no increase in the overall complexity, but at the cost of significant precomputation. As such, it is a form of time/memory trade-off. We have applied this technique to the lightweight cipher KATAN, and significantly improve on previous results on this cipher.

Acknowledgments

The authors would like to thank Henri Gilbert for many helpful discussions, as well as Anne Canteaut and María Naya-Plasencia for their insightful remarks, including the idea of compression (§3.3).

References

1. Martin R. Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.
2. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The Simon and Speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, `http://eprint.iacr.org/2013/404`, 2013.
3. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
4. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Biclique for preimages: attacks on Skein-512 and the SHA-2 family. In Anne Canteaut, editor, *Fast Software Encryption*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.
5. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, Charlotte H. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier, Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450-466. Springer, 2007.
6. Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2011.
7. Christophe de Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.
8. Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. Cryptology ePrint Archive, Report 2013/324, `http://eprint.iacr.org/2013/324`, to appear, 2013.
9. Jian Guo, Thomas Peyrin, Axel Poschmann, Matt Robshaw. The LED Block Cipher. In Bart Preneel, Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326-341. Springer, 2011.
10. Takanori Isobe and Kyoji Shibutani. All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2013.

11. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.
12. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of Trivium and KATAN. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2012.
13. María Naya-Plasencia. How to improve rebound attacks. In Phillip Rogaway, editor, *Advances in Cryptology–CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.
14. Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved meet-in-the-middle cryptanalysis of KTANTAN. In Udaya Parampalli and Philip Hawkes, editors, *Information Security and Privacy*, volume 6812 of *Lecture Notes in Computer Science*, pages 433–438. Springer, 2011.
15. Bo Zhu and Guang Gong. Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64. Cryptology ePrint Archive, Report 2011/619, `http://eprint.iacr.org/2011/619`, 2011.

Appendix: additional details for the attack on KATAN32

• K_1 contains the following 80 round keys (dimension 77):
{20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 93, 94, 96, 97, 98, 100, 104, 107, 113}.
The following keys have a linear contribution: {92, 99, 109, 124}.

• K_2 contains the following 81 round keys (dimension 77):
{213, 215, 217, 219, 221, 223, 225, 227, 229, 231, 233, 235, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305}.
The following keys have a linear contribution: {163, 185, 188, 198}.

• The following 35 round keys are spared by the match box:
{175, 179, 183, 187, 191, 193, 195, 196, 197, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236}.