Differential Meet-In-The-Middle Cryptanalysis

Christina Boura[1], Nicolas David[2], Patrick Derbez[3], Gregor Leander[4], and María Naya-Plasencia[2]

[1] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France
christina.boura@uvsq.fr
[2] Inria, France
{nicolas.david,maria.naya-plasencia}@inria.fr
[3] Univ Rennes, Inria, CNRS, IRISA, France
patrick.derbez@irisa.fr
[4] Ruhr University Bochum, Bochum, Germany
gregor.leander@rub.de

Abstract. In this paper we introduce the differential-meet-in-the-middle framework, a new cryptanalysis technique against symmetric primitives. The idea of this new cryptanalysis method consists in combining into one attack techniques from both meet-in-the-middle and differential cryptanalysis. The introduced technique can be seen as a way of extending meet-in-the-middle attacks and their variants but also as a new way to perform the key recovery part in differential attacks. We provide a simple tool to search, given a differential, for efficient applications of this new attack and apply our approach, in combination with some additional techniques, to `SKINNY-128-384`. Our attack on `SKINNY-128-384` permits to break 25 out of the 56 rounds of this variant and improves by two rounds the previous best known attacks in the single key model.

Keywords: new cryptanalysis family, differential cryptanalysis, meet-in-the-middle cryptanalysis, `SKINNY`

1 Introduction

Since the 1970's and the standarization of the `DES` block cipher, dozens, even hundreds of different symmetric primitives have been designed to address special needs and industrial requirements or to provide an answer to particular research problems. A fundamental procedure that permits to decide which among those primitives can be trusted and safely deployed is cryptanalysis. The first symmetric cryptanalysis techniques were developed in the late 1980's and in the beginning of the 1990's. Among these first attacks, one should cite of course differential [7] and linear cryptanalysis [31], but also boomerang [38] and rectangle attacks [6], impossible differential cryptanalysis [27, 5], higher-order differential cryptanalysis [28], meet-in-the-middle attacks [17] or differential-linear attacks [29]. More attacks appeared in succession to new designs, as for example the square attack [14], particularly well-adapted to `AES`-like constructions or the subspace invariant attacks [30] that worked well against some particular lightweight ciphers. In parallel, some techniques, as the division property [36], permitted to define a new algorithmic framework to generalize older attacks. Nowadays, it is however more and more rare to come up with entirely new cryptanalysis techniques, while improvements of the known ones are more common.

Among the existing cryptanalysis techniques, differential attacks [7] are probably the oldest and the most well studied cryptanalysis methods. Their idea is to exploit an input difference that propagates through the cipher to an output difference with a high probability. Through the years, these attacks have been refined and many improvements to different steps of the attack procedure have been introduced. One can cite the use of structures to build up the plaintext or ciphertext pairs [8], the use of truncated differentials [28], conditional differentials [26], the technique of probabilistic neutral bits [13] or refinements in the key recovery process. A first research question in link with our work is the following:

Question 1. Do there exist alternative methods for doing the key recovery step of a differential attack more efficiently?

Another popular technique that has been useful in many cryptanalysis applications, and the subject of a large number of improvements and further studies is meet-in-the-middle (MITM) cryptanalysis [17]. The idea of basic MITM attacks is to split the cipher into two parts, where each part can be computed with partial knowledge of the key. An attacker can then validate partial key guesses by checking for a match in the middle. Many extensions and refinements of the basic attack exist today. One can for example cite the technique of partial matching, where only a part of the middle state is known, guessing some bits of the internal state [20], the all-subkeys approach [24], the splice-and-cut technique [1, 2, 22] and the sieve-in-the-middle (SITM) approach [11] that permits to extend the length of a MITM attack by searching for a match through an extra S-box layer in the middle. Finally, the method of bicliques is a cryptanalysis technique [9] that aims at extending MITM attacks by some rounds. A second research problem of interest to us and that motivated our initial work is:

Question 2. Is there a new way to permit to a MITM attack to cover more rounds?

In this paper we provide a positive answer to both questions by proposing a new cryptanalysis technique that we call the *differential-meet-in-the-middle attack*. The idea of this new technique is to use a differential to cover several middle rounds of the cipher while running a meet-in-the-middle attack on its external rounds. More precisely, attacks relying on the MITM technique allow to perform in parallel guesses of the initial and final involved keybits.

With our new technique, a middle part of the cipher is covered by a differential and thus we have to apply MITM techniques on much less rounds than with all previous MITM-type attacks. Note that Demirci-Selçuk MITM attacks [16] applied with the differential enumeration technique [19] also combine to some extent truncated differentials and the MITM approach, but in this case the guess is done in parallel on the inner part (with a truncated-based distinguisher) and the external part, trying to match some particular properties of the differential set. Interestingly, our new method can also be interpreted as a differential attack where the key recovery step is done in a different way. Indeed, starting from a given plaintext-ciphertext pair, we guess in parallel the input keys that allow us to compute another plaintext ensuring the given input difference of the differential, and the output keys that allow us to compute the other ciphertext that ensures the output difference of the differential, and compute the associated plaintext for all these ciphertexts by making calls to the oracle. Next, we try to match the list of plaintexts computed with the guesses of the input key bits and the list of plaintexts computed with the list of output keys by finding a collision. We repeat this for enough plaintexts so that we can expect one of them to satisfy the differential. All collisions found will imply a potential candidate for the associated guess of keys. This can usually be done efficiently with list merging algorithms like the ones in [33].

In order to demonstrate the efficiency of our new cryptanalysis technique, we apply it to the block cipher `SKINNY` and we show how to break 25 out of the 56 rounds of the `SKINNY-128-384`, the 128-bit block variant employing a 384-bit key. Our attack improves by two the number of rounds of the previous best attack against this variant in the single key model. A summary of the best attacks against `SKINNY-128-384` is given in Table 1.[5]

Table 1. Best attacks against `SKINNY-128-384` in the single key (SK) model together with the results presented in this paper. ID stands for impossible differentials, MITM for meet-in-the-middle attacks and DS-MITM for Demirci-Selçuk-type MITM.

# Rounds	Data	Time	Memory	Type	Ref.
21	2^{123}	$2^{353.6}$	2^{341}	ID	[39]
21	$2^{122.89}$	$2^{347.35}$	2^{336}	ID	[23]
22	2^{96}	$2^{382.46}$	$2^{330.99}$	DS-MITM	[35]
22	$2^{92.22}$	$2^{373.48}$	$2^{147.22}$	ID	[37]
23	2^{104}	2^{376}	2^{8}	MITM	[18]
23	2^{117}	$2^{361.9}$	$2^{118.5}$	Diff-MITM	Section 5.1
24	2^{117}	$2^{361.9}$	2^{183}	Diff-MITM	Section 5.2
24	$2^{122.3}$	$2^{372.5}$	$2^{123.8}$	Diff-MITM	Section 5.3
25	$2^{122.3}$	$2^{372.5}$	$2^{188.3}$	Diff-MITM	Section 5.3

The rest of the paper is organized as follows. Section 2 describes the general framework of our new cryptanalysis technique, compares it to both differential and MITM attacks and provides several improvement techniques. Section 3

[5] Note that [23] provides a 26-round integral attack against `SKINNY-128-384` but it relies on differences in the tweak.

describes a new tool for searching attacks of this type on a given cipher. The specifications of the SKINNY tweakable block cipher are given in Section 4. Our attacks against SKINNY-128-384 are described in Section 5 and finally several open problems are discussed in Section 6.

2 The new attack: Differential MITM

We propose in this work a new cryptanalysis technique against symmetric primitives. This new attack aims at combining meet-in-the-middle (MITM) attacks together with differential cryptanalysis, and we will call this new technique *differential-MITM*. The main motivation of our work was to investigate whether there exists a method for reaching more rounds than the sieve-in-the-middle attack [11, 12], an extension of classical MITM attacks. However, our technique can also be interpreted as a new key-recovery method to apply in differential cryptanalysis. We will present in this section a high-level description of the new technique. More precisely, we will provide a general framework that describes how to mount a differential-MITM attack in a generic and simple way, and we will show how to combine this generic method with two techniques: the parallel treatment of data partitions in order to add one round mostly for free, as well as a technique to reduce the data complexity.

2.1 General framework

Consider an n-bit cipher E decomposed into three sub-ciphers: $E_{out} \circ E_m \circ E_{in}$, as depicted in Figure 1. Let the number of rounds of E_{in}, E_m and E_{out} be r_{in}, r_m and r_{out} respectively. Finally, let Δ_x be the input difference to the middle part E_m, Δ_y the output difference of E_m and suppose that the differential $\Delta_x \rightarrow \Delta_y$, covering the r_m middle rounds, has probability 2^{-p}.

We start our analysis with a first randomly chosen plaintext P and its associated ciphertext C, and we aim at generating a second plaintext-ciphertext pair $(\widetilde{P}, \widetilde{C})$ such that together they satisfy the differential on the middle rounds. Our new idea is to generate $(\widetilde{P}, \widetilde{C})$ with a meet-in-the-middle approach. For this, candidate plaintexts $\widetilde{P}$ are computed from both the plaintext P and the difference Δ_x while candidate ciphertexts $\widetilde{C}$ are computed from C and Δ_y. The match is then performed on the relation $E(\widetilde{P}) = \widetilde{C}$ (or $\widetilde{P} = E^{-1}(\widetilde{C})$).

Note that the roles of the upper and lower part can be interchanged without loss of generality in order to optimize the data and memory complexity, if we consider that the access to both the encryption and the decryption oracles is granted.

Upper part. Given P, the aim is to guess the minimal amount of key information, that we will denote by k_{in}, such that we can compute the associated $\widetilde{P}$ that ensures $E_{in}(P) \oplus E_{in}(\widetilde{P}) = \Delta_x$ if the guess of k_{in} corresponds to the secret key. For each guess i for k_{in}, we obtain a different candidate for $\widetilde{P}$, that we denote by $\widetilde{P}^i$, leading to a total of $2^{|k_{in}|}$ such values. From them, we can compute the

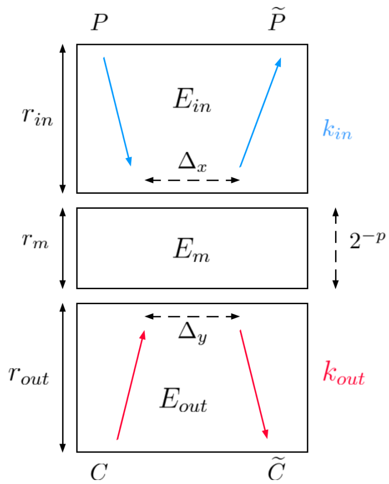


Fig. 1. A high-level description of the Differential - MITM technique

$2^{|k_{in}|}$ associated ciphertexts $\widehat{C}^i = E(\widetilde{P}^i)$ with calls to the encryption oracle and store them in a hash table H.

Lower part. Similarly, given C, we can guess some key material k_{out}, of bit-length $|k_{out}|$, and compute a new ciphertext $\widetilde{C}$ that satisfies the equation $E_{out}^{-1}(C) \oplus E_{out}^{-1}(\widetilde{C}) = \Delta_y$ if the key guess is correct. We obtain $2^{|k_{out}|}$ values for $\widetilde{C}^j$, each associated to a guess j for k_{out}.

Number of pairs and match. For the correct key guess, the transition $\Delta_x \rightarrow \Delta_y$ will happen with a probability 2^{-p}. Therefore, we will repeat the upper and lower procedures 2^p times with 2^p different messages P_ℓ so that we can expect one pair $(P_\ell, \widetilde{P}_\ell^i)$ to satisfy the differential together with the associated pair $(C_\ell, \widetilde{C}_\ell^j)$. When this is the case, we will find a collision for a certain ℓ between a $\widehat{C}_\ell^i$ computed in the upper part and stored in H and a $\widetilde{C}_\ell^j$ computed from the lower part. Each collision (i, j) has an associated key guess $k_{in} = i, k_{out} = j$, that we will consider as a potential candidate. The number of expected collisions for each fixed P_ℓ is $2^{|k_{in}|+|k_{out}|-|k_{in} \cap k_{out}|-n}$.

Complexity. The time complexity of this attack can be estimated as

$$\mathcal{T} = 2^p \times (2^{|k_{in}|} + 2^{|k_{out}|}) + 2^{|k_{in}|+|k_{out}|-|k_{in} \cap k_{out}|-n+p},$$

where the first term corresponds to the computations done in E_{in} and E_{out}, and the last one to the number of expected key candidates. With this, we recover $k_{in} \cup k_{out}$, so if we expect fewer key candidates than the whole set $k_{in} \cup k_{out}$, (i.e $|k_{in}| + |k_{out}| - |k_{in} \cap k_{out}| - n + p < |k_{in} \cup k_{out}|$, which holds as long as $p < n$), we can guess the remaining bits of the master key and test the guess

Algorithm 1 Differential MITM attack

```
while right key not found do                          ▷ 2^p trials expected
    Randomly pick P
    C ← E(P)                                          ▷ Oracle call
    H ← ∅                                             ▷ hash table initialisation
    for each guess i for k_in do                      ▷ Forward computation
        Compute P̃^i from i and P
        Ĉ^i ← E(P̃^i)                                  ▷ Oracle call
        H[Ĉ^i] ← H[Ĉ^i] ∪ {i}
    end for
    for each guess j for k_out do                     ▷ Backward computation
        Compute C̃^j from j and C
        for each i ∈ H[C̃^j] do
            Complete (i, j) to retrieve the master key
            Try candidates against extra data
        end for
    end for
end while
```

with additional pairs. Thus we recover the whole key with a complexity smaller than the cost of an exhaustive key search, and an additional cost of

$$2^{k-(|k_{in} \cup k_{out}|)} \times \max\{1, 2^{|k_{in}|+|k_{out}|-|k_{in} \cap k_{out}|-n+p}\}$$

to be added to the time complexity $\mathcal{T}$. In the expected case where $|k_{in}|+|k_{out}|-|k_{in} \cap k_{out}|-n+p \geq 0$, the total time complexity is thus

$$\mathcal{T} = 2^p \times (2^{|k_{in}|} + 2^{|k_{out}|}) + 2^{|k_{in} \cup k_{out}|-n+p} + 2^{k-n+p}.$$

The (naive) data complexity of this first version of the attack can be estimated as

$$\mathcal{D} = \min(2^n, 2^{p+\min(|k_{in}|,|k_{out}|)}).$$

Finally, the naive memory complexity is given by $\mathcal{M} = 2^{\min(|k_{in}|,|k_{out}|)}$, though it can be improved to $2^{\min(|k_{in}|-|k_{in} \cap k_{out}|, k_{out}-|k_{in} \cap k_{out}|)}$ by first guessing the common key material before running the attack.

2.2 Improvement: Parallel partitions for layers with partial subkeys

We will show now that in the case where the round key addition does not affect the whole state but only $m < n$ bits of it (as is for instance the case in Feistel constructions [21], or in the `SKINNY` [4] and `GIFT` [3] ciphers), we can add one additional round to the attack. If $p > m$, the time complexity of adding this round will not be affected. The exact data complexity of the attack must however be checked case-by-case, as it might depend on the configuration of the differences in the external states, but a technique proposed in the next subsection can allow to reduce it. The memory complexity will be a priori increased.

The main idea here is to consider, in addition of guessing the upper and lower key bits in parallel, a partial guess of the starting states in parallel, so that 2^m states from the penultimate state and the last state (or the first state, without loss of generality) are guessed in parallel, without needing to guess the key that allows the transition from one to the other. When performing the final match, we will take into account this key transition. Actually, it can be seen as considering 2^m plaintexts P_i and ciphertexts C_j in parallel, without knowing which ones would match together, as this transition will be determined by the last round-key.

Since we expect to try on average 2^p plaintexts in order to find one that will satisfy the differential of probability 2^{-p}, we can divide the final state of size 2^n into two parts. The part without the key addition will take 2^{p-m} different values, and the attack will be repeated for each one of those.

On the other hand, the part affected by the key addition, will take 2^m possible values for X, the state before the key addition, and for each we can compute the state S_{r-1} in Figure 2 from the output of the differential MITM attack. From this state, we will guess the k_{out} bits, in order to compute the associated state potentially generating the output difference of the middle differential, obtaining $2^{|k_{out}|+m}$ candidates to match. In parallel, the state Y after the key addition will also take all the 2^m possibilities, and with them we decrypt in order to obtain the plaintext, and do the upper key guessing procedure to deduce the good pairs, obtaining $2^{|k_{in}|+m}$ candidates. The number of possible solutions might seem higher by a factor of 2^{2m}, but note that we have to match X and Y, as well as their associated pairs X' and Y', and they must satisfy $X \oplus X' = Y \oplus Y'$. This adds m bit-conditions, or more if this final subkey was already determined by k_{in} and k_{out}, which is usually the case. This implies m additional conditions, and $2^{2m}2^{-m}2^{-m} = 1$, so the cost, given by the number of solutions, stays exactly the same as the attack with one round less. We will see how this technique can be applied in practice in Section 5.

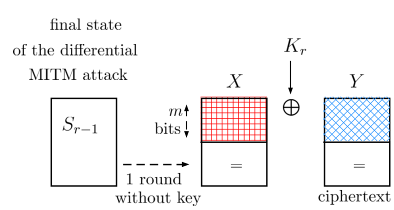


Fig. 2. Partial guess of the final state to add one round for free. S_{r-1} is the final state of the simple differential MITM attack.

2.3 Improvement: Reducing data with imposed conditions

We explain here a way to obtain a time-data-memory trade-offs for the original attack. If when choosing the plaintext P, we force x of its bits, that might have been active otherwise, to a certain value, and if we expect the same from the associated plaintext $\widetilde{P}$, the overall probability of the attack will decrease to 2^{-p-x}, as we will have to repeat the procedure until a $\widetilde{P}$ that satisfies this constraint is found. More precisely, if $\widetilde{P}$ does not fit this condition, the corresponding tuple will not be stored in the hash table since we do not have access to its ciphertext. However by doing so, the data complexity will be reduced by a factor of 2^x as well as the memory complexity. When combining this technique with the previous one, we can derive the following two inequalities for x:

$$p + x \leq n - x \quad \text{and} \quad 2^{p+x}(2^{|k_{in}|} + 2^{|k_{out}|}) < 2^k.$$

This type of trade-off applies in particular when all the code book, 2^n, would be needed before fixing the x bits, and the data complexity becomes 2^{n-x}.

Data reduction without time increase As the total number of candidates for the key of the input part (respectively output) will be $2^{|k_{in}|-x}$ (respectively $2^{|k_{out}|-x}$), if we are able to find these candidates with their associated $\widetilde{P}$ (respectively $\widetilde{C}$) in a complexity given by the number of solutions, the time complexity would become:

$$2^{p+x}(2^{|k_{in}|-x} + 2^{|k_{out}|-x}) = 2^p(2^{|k_{in}|} + 2^{|k_{out}|}),$$

which allows us to reduce the data complexity to 2^{n-x} while not increasing the time complexity. The optimal data complexity in this case will be $2^{\frac{n+p}{2}}$, obtained with x equals to $\frac{n-p}{2}$.

This can actually be done in many cases using rebound-like techniques [32]. This is the case of all of our attacks summarized in Table 1.An example can be seen in Section 5.

2.4 Discussion and Comparison

As argued before, our new cryptanalysis technique is closely related to two families of cryptanalysis: MITM attacks and differential attacks. In this section we will discuss similarities and differences between these families and will try to identify cases where our new technique might be efficient or cases where it permits to reach better results compared to the best known attacks.

Relation to MITM attacks In relation to MITM attacks and its variants, like the sieve-in-the-middle technique, our attack, already if using a differential with probability one, could have, a priory, the potential of reaching more rounds. The starting point of our research was whether it was possible to add even more rounds in the middle of a MITM-like attack and this is how we came up with the new attack. The data complexity of the new attack could be higher than a classical MITM one as now we compute a new $\widetilde{P}$ from each guess of the key, and this plaintext can take many different values, besides the 2^p different plaintexts P_ℓ taken as starting points in order to find one that satisfies the differential. On the other hand, despite the fact that the sets of bits k_{in} or k_{out} involved in the parallel computations of the differential MITM attack are not determined in the same way as the key bits involved in MITM attacks, we expect those quantities to be relatively close under similar settings, as this principally depends on the propagation properties of the round function. More precisely, it seems that more aligned [10] the round function is, closer the sets will be.

Therefore, we expect that ciphers where classical MITM attacks work well, can also be interesting targets for differential-MITM attacks. This is actually how we found the application shown in Section 5 on `SKINNY`. Indeed, the best known attack against `SKINNY-384-128` previous to ours was a MITM one [18].

Relation to differential attacks Curiously enough, our new attack can also be seen as a new way of performing the key-recovery part associated to a differential distinguisher.

Classical differential attacks. A differential attack starts with a differential distinguisher on r_m rounds of relatively high probability 2^{-p}. One then typically extends the distinguisher by r_{in} rounds to reach the plaintext state and by r_{out} rounds to reach the ciphertext state, with probability 1 as depicted in Figure 3. Structures are then used to build plaintext (or ciphertext) pairs. A structure of size 2^s allows to build 2^{2s-1} pairs (though building all these pairs is rarely needed), but for each structure we need to consider typically an additional probability so that the pairs from the structure satisfy the input difference of the distinguisher. If we are not considering truncated but fixed differentials, as it will be our case, the probability of reaching a fixed difference Δ_x is approximately 2^{-s}. In order to expect that at least one pair will satisfy the distinguisher, we will need to consider 2^{p-s+1} structures, each of size 2^s, and for each structure, 2^{s-1} pairs will reach the input difference Δ_x, leading to a total of $2^{p-s+1+s-1} = 2^p$ pairs reaching Δ_x. Typically, for determining a priori potential good pairs, one performs some sieving that will allow not to try all the pairs from all the structures. This sieving is done by looking at the activity pattern of the ciphertexts and can be estimated as $2^{-c} = 2^{-n+a}$, where a is the bit-size of the active part in the ciphertext. The final key recovery part depends on the particular properties of the round function, but its complexity can be considered to be relatively small thanks to early abort and divide-and-conquer techniques.

The data complexity of such an attack is 2^{p+1} and the time complexity can be estimated as

$$2^{p+1} + 2^{p-s+1}2^{2s-1}2^{-c}C_k \approx 2^{p+s-n+a},$$

where C_k is the average cost of determining the key bits for each candidate pair.

We can see that if s and a are big enough, i.e. $a + s >> n$, which can happen when several rounds are appended, the complexity of a differential MITM

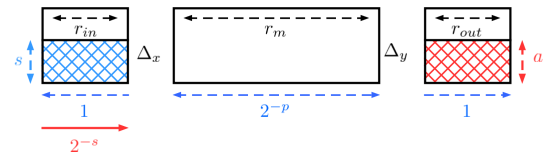


Fig. 3. Framework of a classical differential attack.

attack for an equivalent number of rounds might be more interesting. Indeed, the influence of the input and output extensions to the complexity are added and not multiplied. In particular, our attack can become much more efficient when the key size of the cipher is bigger than the state size, otherwise 2^p might already be close to the limit.

3 Automatic detection of involved keys

In many cases, it is technically very easy to exactly determine which information about the key is required in the forward or backward computation. As often, when it comes to the question of dependency only, the easiest and less error-prone method is to experimentally determine which bits have an actual influence. Assume we are given (the implementation of) a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and want to determine if the ith input bit of x has an influence on the output of $F(x)$, that is if there exist an input x such that

$$F(x) \neq F(x \oplus e_i),$$

where e_i is the vector that has a one exactly at position i.

For this, we could simply take a random input x and compute $F(x) \oplus F(x \oplus e_i)$. If the result is non-zero, we know that the output depends on the ith input bit. After repeating this process a few times and if we always get zero as a result, we conclude that the ith bit does not have an influence on the output. The later decision might of course be wrong (while the former never is). However, for our applications this is (i) very unlikely to happen due to the construction of the round functions and (ii) irrelevant for the attacks as a key bit that influences the output only in one out of many outputs usually does not have to be guessed.

Focusing on our target, given the implementation of the cipher, i.e. in particular the (round-reduced) encryption and decryption procedures along with the key schedule, we can easily process as represented in Algorithm 2.

The nice feature of the algorithm is that it works for any cipher structure and without the need to know or implement any internal details. However, on this generality we might miss many possible improvements. As an example, consider the key schedule of `SKINNY`. Here, every round-tweak-key bit is the sum of three bits of (updated) master tweak-key bits. The algorithm above would (correctly) detect the dependence of the output on those three bits, but obviously guessing the sum of the bits is enough. We can easily adopt the above algorithm to take

Algorithm 2 Check Dependencies

Input: An Implementation of the round reduced encryption and decryption E_r, a difference Δ

Output: The set of key bits $\mathcal{K}$ required to propagate Δ through E_r^{-1}

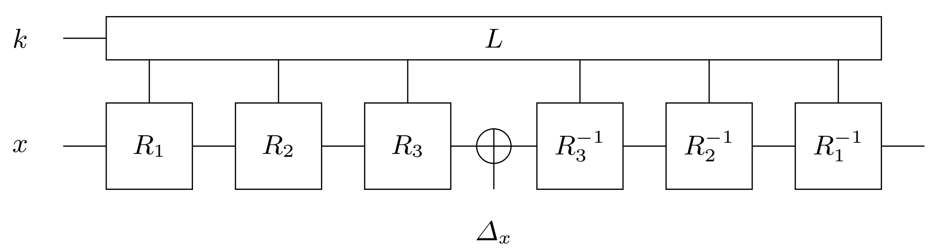
$\mathcal{K} \leftarrow \emptyset$
for each i from 0 to $n-1$ **do**
  for each t from 1 to tries **do**
    $x \leftarrow$ random message
    $k \leftarrow$ random key
    $y_1 \leftarrow E_r(k, x)$ ▷ Query Encryption
    $z_1 \leftarrow E_r^{-1}(k, y_1 \oplus \Delta)$ ▷ Query Decryption
    $y_2 \leftarrow E_r(k \oplus e_i, x)$ ▷ Query Encryption with bit flipped
    $z_2 \leftarrow E_r^{-1}(k \oplus e_i, y_2 \oplus \Delta)$ ▷ Query Decryption with bit flipped
    if $z_1 \neq z_2$ **then**
      Include key bit i to $\mathcal{K}$
    end if
  end for
end for
Return $\mathcal{K}$

into account linear (tweak) key-schedules. The main idea is not to flip master key-bits directly, but rather round-key bits.

Let us denote the linear key schedule by $L : \mathbb{F}_2^{\kappa} \to \mathbb{F}_2^{n\dot{r}}$. The ith bit of the expanded key can thus be written as $\langle L_i, k \rangle$ where L_i denotes the ith row of the matrix corresponding to L. Furthermore we denote by $\widehat{E}_r$ the encryption excluding the key schedule, i.e.

$$E_r(k, x) = \widehat{E}_r(L(k), x).$$

Instead of master key-bits, we now aim at computing the round-key bits that the encryption depends on and collecting the corresponding linear combinations of the master-key.



The vector-space contains the information that is sufficient to guess in order to compute the upper part of the attack. The dimension of $\mathcal{K}$ corresponds to the

Algorithm 3 CHECK LINEAR-DEPENDENCIES

Input: An Implementation of the round reduced encryption and decryption $\widehat{E}_r$, the key-schedule L and a difference Δ

Output: A linear subspace of key bits $\mathcal{K}$ required to propagate Δ through $\widehat{E}_r^{-1}$

```
K ← {}
for each i from 0 to n...r − 1 do
    for each t from 1 to tries do
        x ← random message
        k ← random key
        y1 ← Ê_r(L(k), x)                      ▷ Query Encryption
        z1 ← Ê_r^{-1}(L(k), y1 ⊕ Δ)            ▷ Query Decryption
        y2 ← Ê_r(L(k) ⊕ e_i, x)                ▷ Query Encryption with bit flipped
        z2 ← Ê_r^{-1}(L(k) ⊕ e_i, y2 ⊕ Δ)      ▷ Query Decryption with bit flipped
        if z1 ≠ z2 then
            Include L_i to K
        end if
    end for
end for
Return span(K)                 ▷ Vector space spanned by the corresponding L_i
```

amount of information that has to be guessed, its Gauss-Jordan-basis contains information on which master key bits can be guessed equivalently. This algorithm again is easy to adapt given an implementation of a cipher. In practice, given a differential $\Delta_x \to \Delta_y$, we can apply this algorithm on both (E_{in}, Δ_x) and (E_{out}^{-1}, Δ_y) to respectively get the sets k_{in} and k_{out} required in our new framework.

For SKINNY, Algorithm 3 allows to complete both the differential trails given in Section 5 into attacks against 23 and 24 rounds, that we will finally extend by one additional round at the end to get the currently best known results on SKINNY-128-384.

4 The SKINNY family of ciphers

The SKINNY family of tweakable block ciphers was designed by Beierle et al. [4]. It is a family of lightweight ciphers following a classical SPN structure but implementing a very compact S-box, a linear layer based on a sparse non-MDS binary matrix and a lightweight key schedule. The block size n can be 64 or 128 bits and for both versions the state is seen as a 4×4 matrix of 4-bit or 8-bit cells. Row 0 is considered the uppermost one and column 0 is taken to be the leftmost one. The numbering of the words inside the state matrix is as follows.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

SKINNY follows the tweakey framework [25] and XORs a tweakey to the two upmost rows of the state. There exist three main variants for the tweakey size: $t = n$, $t = 2n$ and $t = 3n$ and the corresponding variant is denoted by SKINNY-n-t. Furthermore, the tweakey to block size ratio is denoted by $z = t/n$. The tweakey state is viewed also as a set of z 4×4 arrays of cells. For $z = 3$, which is the variant of interest to us, the three tweakey arrays are denoted by **TK1**, **TK2** and **TK3**.

The round function of SKINNY is depicted on Figure 4, and the number of times this function is iterated depends on both n and t, as shown in Table 2.

Table 2. Number of rounds for each of the main variants SKINNY-n-t.

Block size n	$t = n$	$t = 2n$	$t = 3n$
64	32	36	40
128	40	48	56

One round of SKINNY is composed of five operations applied in the following order: SubCells (SC), AddConstants (AC), AddRoundTweakey (ART), ShiftRows (SR) and MixColumns (MC). We know briefly describe the operations that are of interest to us.

SubCells (SC) This operation applies an S-box to all cells of the state. The table representation of the 8-bit S-box used in the 128-bit variants is given in Appendix A.

AddRoundTweakey (ART) We describe this step only for the variants with $z = 3$. Here, the first and second rows of the 3 tweakey arrays **TK1**, **TK2** and **TK3** are extracted and XORed to the internal state, respecting the bit positions inside the arrays. More formally, if $\mathtt{IS}_{i,j}$ is the cell at the intersection of row i and j of the state, we have that for $(i, j) \in \{0, 1\} \times \{0, 1, 2, 3\}$:

$$\mathtt{IS}_{i,j} = \mathbf{TK1}_{i,j} \oplus \mathbf{TK2}_{i,j} \oplus \mathbf{TK3}_{i,j}.$$

ShiftRows (SR) This operation rotates the cells inside a row to the right by a certain offset that depends on the row. More precisely, cells in row i, where $0 \leq i < 4$, are rotated by i positions to the right.

MixColumns (MC) This operation updates the state by multiplying each column by a binary matrix $\mathbf{M}$. This matrix, as well as its inverse are as follows.

$$\mathbf{M} = \begin{pmatrix} 1&0&1&1 \\ 1&0&0&0 \\ 0&1&1&0 \\ 1&0&1&0 \end{pmatrix}, \quad \mathbf{M^{-1}} = \begin{pmatrix} 0&1&0&0 \\ 0&1&1&1 \\ 0&1&0&1 \\ 1&0&0&1 \end{pmatrix}.$$

Next we describe the tweakey schedule of SKINNY-128-384, the variant we analyse in this work.

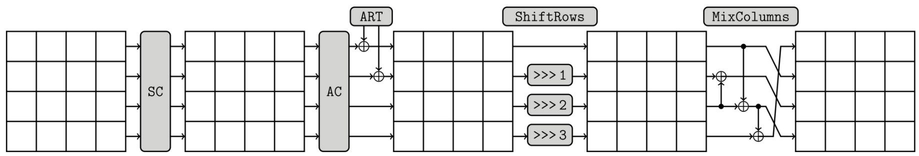


Fig. 4. Round function of SKINNY [4]

Tweakey schedule of SKINNY-128-384 At each round, all tweakey arrays are updated as follows (see also Figure 5). First, the same permutation P_T is applied on the cell positions of the 3 tweakey arrays: for all $0 \leq i \leq 15$, we have that $\mathbf{TK1}_i \leftarrow \mathbf{TK1}_{PT[i]}$ with

$$P_T = [\mathtt{9}, \mathtt{15}, \mathtt{8}, \mathtt{13}, \mathtt{10}, \mathtt{14}, \mathtt{12}, \mathtt{11}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}],$$

and the same exact permutation is applied to the cells of **TK2** and **TK3**. Finally, the cells of the first two rows of **TK2** and **TK3** are individually updated by the LFSR :

$$(\mathtt{x}_7||\mathtt{x}_6||\mathtt{x}_5||\mathtt{x}_4||\mathtt{x}_3||\mathtt{x}_2||\mathtt{x}_1||\mathtt{x}_0) \rightarrow (\mathtt{x}_0 \oplus \mathtt{x}_6||\mathtt{x}_7||\mathtt{x}_6||\mathtt{x}_5||\mathtt{x}_4||\mathtt{x}_3||\mathtt{x}_2||\mathtt{x}_1),$$

where x_0 is the LSB in a byte.

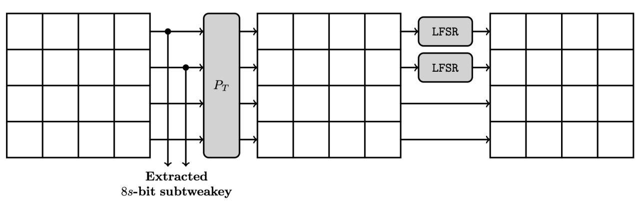


Fig. 5. Tweakey schedule of SKINNY. All tweakey arrays **TK1**, **TK2** and **TK3** follow the same transformation with the only exception that no LFSR is applied to **TK1**. [4]

Property. In most attacks, including ours, it is more efficient to guess round-key bits than key bits. Since the key-schedule of SKINNY is fully linear, guessing enough (e.g. 384 in the **TK3** model) independent round-key bits allows to uniquely determine the master key and thus all remaining round-key bits. SKINNY has a unique property which makes easy the evaluation of the dimension of any set of round-key bytes: in the **TKx** model, a round-key byte $\mathtt{K}_\mathtt{r}[\mathtt{i}]$ always depends on exactly x master key bytes, one from each **TK** keys and they all have the same index. Furthermore, given x round-key bytes from the 30 first rounds and depending on the same x master key bytes, they are always independent and allow to uniquely determine their x corresponding master key bytes.

5 Differential-Meet-the-Middle attacks against `SKINNY-128-384`

We present now three attacks against round-reduced variants of `SKINNY-128-384` by using our new differential-meet-in-the-middle technique. We first describe a simple attack against 23 rounds as well as several improvements and trade-offs. We then explain how this attack can be extended to an attack on 24 rounds without increasing the attack's overall complexity. Finally, we describe a new attack against 25 rounds.

5.1 An attack against 23-round `SKINNY-128-384`

As explained in Section 2, differential-meet-in-the-middle attacks rely on two classical cryptanalysis techniques: differential attacks and meet-in-the-middle attacks. The main idea is to use a meet-in-the-middle attack to generate a pair following a given differential in the middle rounds. Thanks to this procedure, we are able to extend a differential distinguisher by more rounds than with a classical early-abort procedure, as discussed in Section 2.4.

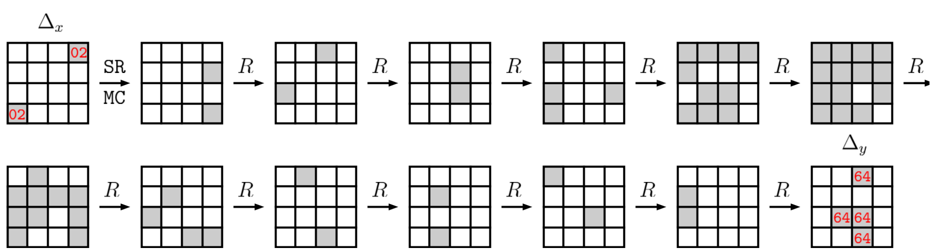


Fig. 6. Truncated differential trail for the attack on 23 rounds.

Differential. The truncated differential used in our attack against 23 rounds of `SKINNY-128-384` is depicted in Figure 6. It has 56 active S-boxes, without counting those of the first and the last round. We verified that this truncated differential can be successfully instantiated by using the constrained programming `Choco`-solver [34], and more precisely the model developed by Delaune *et al.* to search for the best differential characteristics for the `SKINNY` family of block ciphers [15]. The best instantiation of this truncated differential has a probability of 2^{-119} and there are in total 2048 instantiations with this same probability. These instantiations can be divided into four groups $(\Delta_x^{(i)}, \Delta_y^{(i)})$, for $i = 1, 2, 3, 4$, each one having 512 trails inside, starting with the same difference $\Delta_x^{(i)}$ and terminating after 13 rounds with the same difference $\Delta_y^{(i)}$. We found as well many more differential trails with the same input/output differences but smaller probabilities: 2560 with probability 2^{-120}, 7168 with probability 2^{-121}, 18432 with probability 2^{-122} and 44800 with probability 2^{-123}. Thus the probability of the differential depicted on Figure 6 is higher than $2^{-105.9}$.

The attack. We describe now our core attack against 23-round `SKINNY-128-384`.

1. Ask for the encryption of the whole codebook.
2. Randomly pick one plaintext/ciphertext pair (P, C).
3. For each possible value i of k_{in} compute the tuple $(P, \widetilde{P}, i)$ so that the difference on state after the 6th S-box layer is `[0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0]` (see Figure 6). Doing so requires to know the values of all the active S-boxes involved in the probability 1 transition $\Delta_x \rightarrow \Delta_P$, where Δ_P is the plaintext difference, and k_{in} is the set of round-key bytes needed to compute them from the plaintext.
4. Store all these tuples in a hash table. This step requires to guess 31 round-key bytes as depicted in Figure 7.
5. Similarly, for each possible value j of k_{out} compute the tuple $(C, \widetilde{C}, j)$ so that the difference on the state before the 19th S-box layer is `0x64` on all active bytes. The set k_{out} involves 32 bytes and thus there are 2^{256} such tuples.
6. For each of them check for possible matches on the hash table. The match is performed on both the new ciphertext (i.e. $(\widetilde{P}, \widetilde{C})$ must be a valid plaintext/ciphertext pair) as well as on the linear relations between the round-key bytes of the upper and lower guess.
7. Each match leads to a (full) key candidate that can be tried against very few additional plaintexts (3 in our case).
8. Repeat from Step 2 until the right key is retrieved.

Table 3. Round-key bytes involved in the 23-round core attack.

Byte	k_{in}	k_{out}	# equations
0	$K_0[0]$, $K_2[2]$	$K_{18}[2]$, $K_{20}[4]$, $K_{22}[6]$	2
1	$K_0[1]$, $K_2[0]$	$K_{20}[2]$, $K_{22}[4]$	1
2	$K_0[2]$, $K_2[4]$	$K_{20}[6]$, $K_{22}[5]$	1
3	$K_0[3]$, $K_2[7]$	$K_{20}[1]$, $K_{22}[0]$	1
4	$K_0[4]$, $K_2[6]$	$K_{20}[5]$, $K_{22}[3]$	1
5	$K_0[5]$, $K_2[3]$	$K_{20}[7]$, $K_{22}[1]$	1
6	$K_0[6]$, $K_2[5]$, $K_4[3]$	$K_{20}[3]$, $K_{22}[7]$	2
7	$K_0[7]$, $K_2[1]$, $K_4[0]$	$K_{20}[0]$, $K_{22}[2]$	2
8	$K_1[2]$, $K_3[4]$	$K_{19}[4]$, $K_{21}[6]$	1
9	$K_1[0]$	$K_{19}[2]$, $K_{21}[4]$	0
10	$K_1[4]$	$K_{19}[6]$, $K_{21}[5]$	0
11	$K_1[7]$, $K_3[1]$	$K_{19}[1]$, $K_{21}[0]$	1
12	$K_1[6]$, $K_3[5]$	$K_{21}[3]$	0
13	$K_1[3]$	$K_{19}[7]$, $K_{21}[1]$	0
14	$K_1[5]$, $K_3[3]$	$K_{19}[3]$, $K_{21}[7]$	1
15	$K_1[1]$, $K_3[0]$	$K_{19}[0]$, $K_{21}[2]$	1

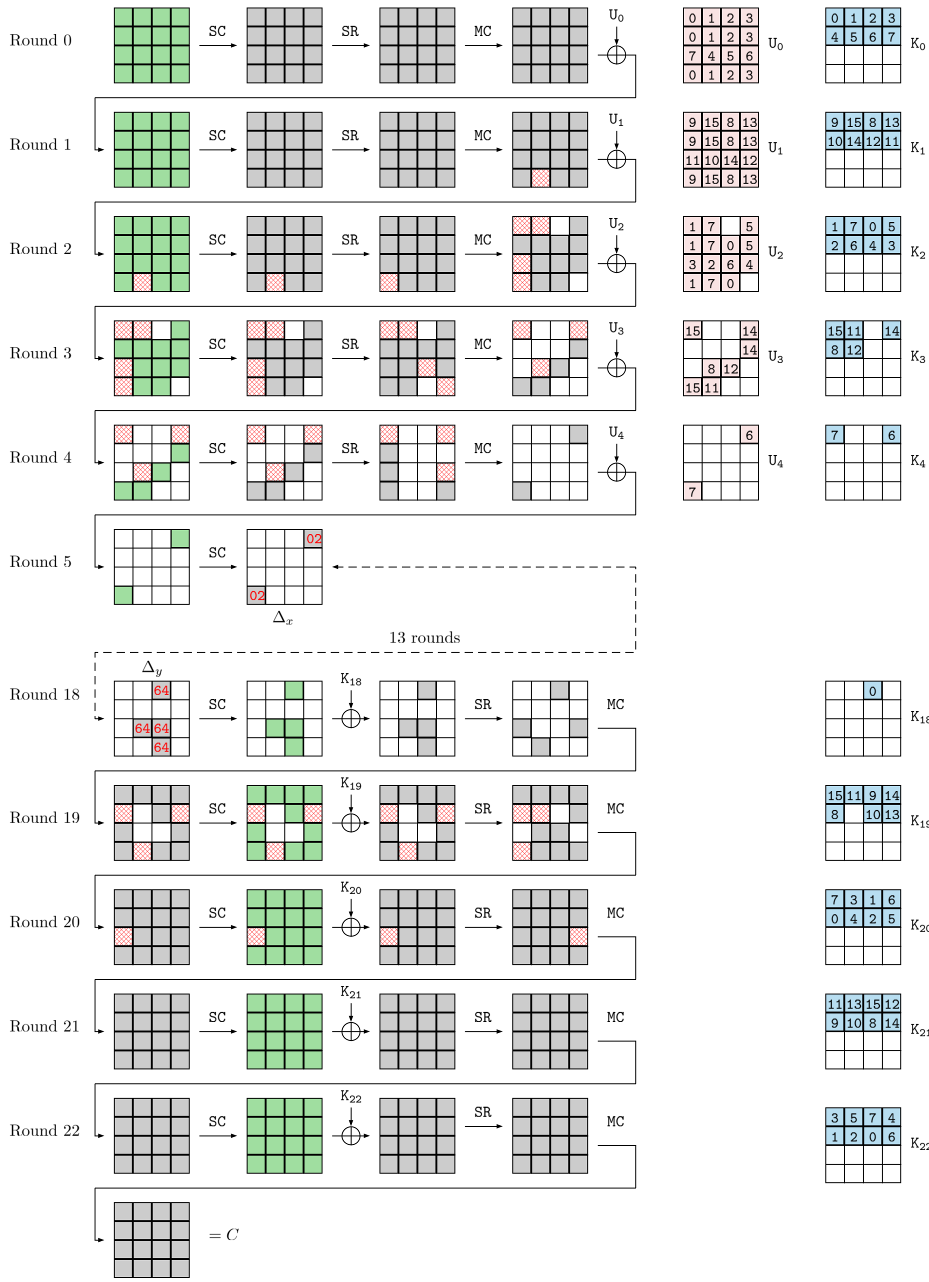


Fig. 7. Core attack against 23 rounds of SKINNY-128/384. Knowledge of blue key bytes allows to compute values of green ones and thus to propagate differences. No difference in both white and red bytes, but the red ones are required to compute green bytes. Indexes in round-key bytes are the indexes of the corresponding master key bytes. The equivalent subkeys $\mathtt{U_i}$ are computed as $\mathtt{MC(SR(K_i))}$, from the original subkeys $\mathtt{K_i}$.

The data complexity of this attack is 2^{128} since in Step 1 we ask for the encryption of the full codebook. The memory complexity is determined by Step 3 in which 2^{248} words of $128+248 = 376$ bits each are stored. Note that we do not need to store C_1 since it is common to all tuples. Thus the memory complexity is $2^{249.5}$ 128-bit words. The time complexity is 2^{248} for computing the hash table, 2^{256} for performing Step 4, and, as shown in Table 3, $k_{in} \cup k_{out}$ is the full key so that the complexity of Step 6 is $2^{384-128} = 2^{256}$. Finally, the attack has to be repeated $2^{105.9}$ times (the probability of the distinguisher) in order to construct one right differential pair. Hence, the overall complexity of our attack is $2^{105.9} \times 2^{256} = 2^{361.9}$.

Decreasing memory complexity. It is possible to decrease the memory complexity of the attack by avoiding the match on the linear relations between both k_{in} and k_{out}. Indeed, since the key-schedule of `SKINNY` is fully linear, we can first guess the intersection of k_{in} and k_{out}, and only then run the attack. The dimension of the intersection is $248 + 256 - 384 = 120$ and thus the memory complexity can be decreased to $2^{249.5-120} = 2^{129.5}$.

Data/Time/Memory trade-off. To decrease the data complexity of our attack, as described in Section 2.3, it is possible to only ask for the encryption of a portion of the whole codebook, let say 2^{128-x} plaintext/ciphertext pairs. In this case, the probability that we have access to the corresponding ciphertext of P_2 is 2^{-x} and the attack has to be ran 2^x times to compensate. Overall, the complexity of our 23-round attack is then $\mathcal{D} = 2^{128-x}$ plaintext/ciphertext pairs, $\mathcal{M} = 2^{129.5-x}$ 128-bit words and $\mathcal{T} = 2^{361.9+x}$ encryptions. To expect at least one pair following the differential under the extra constraint on the plaintexts, x cannot be higher than $(128 - 105.9)/2 = 11.05$.

As explained in Section 2.3, in practice, given any pair of ciphertexts, we can enumerate the possible values for both k_{in} and k_{out} with a complexity roughly equivalent to the number of solutions. Such a procedure is described in Appendix B. Thus, the trade-off does not increase the time complexity and the overall complexity of our attack is $D = 2^{117}$, $T = 2^{361.9}$ and $M = 2^{118.5}$.

5.2 Extension to 24 rounds

Our differential-meet-in-the-middle attack against 23-round `SKINNY` can be extended to an attack against 24 rounds without increasing its overall complexity by using the generic improvement presented in Section 2.2. This can be achieved since on one hand the key-schedule is linear (and enough round-key bytes are involved in our attack so that the master key is fully retrieved) and on the other hand the round-key is only applied on half of the state in each round. The scenario of this new attack is quite similar to the original one:

1. Ask for the encryption of the whole codebook.
2. Pick 2^{64} plaintext/ciphertext pairs (P_ℓ, C_ℓ) such that $\mathtt{MC}^{-1}(C_\ell)$ is constant on the two last rows as depicted in Figure 8. Here we exploit the fact that the round key is only applied on the first two rows of the internal state.

3. As for this original attack, compute all possible tuples $(P_\ell, \widetilde{P}_\ell^i, i)$ for each value i of k_{in} and each P_ℓ from the structure defined at the previous step such that the state difference after the 6th S-box layer is `0x02` on both active bytes.
4. Store them in a hash table. Note that the tuples are computed for all the 2^{64} plaintexts selected at Step 2 so the memory complexity is $2^{248+64} = 2^{312}$ 504-bit words.
5. For each value j of k_{out} and each state $S_{23,\ell}$ coherent with Step 2 (*i.e.* 2^{64} states, one for each possible value of the round-key K_{23}), compute all possible tuples $(S_{23,\ell}, \widetilde{S}_{23,\ell}^j, j)$ so that the difference on state before the 19th S-box layer is `0x64` on the four active bytes.
6. Check for possible matches on the hash table. The match is now performed on three quantities:
   – the difference between the last states: $C \oplus \widetilde{C} = \mathtt{MC} \circ \mathtt{SR}(\mathtt{SC}(S_{23}) \oplus \mathtt{SC}(\widetilde{S}_{23}))$. This is a 64-bit filter because the difference is zero on the two last rows since $\mathtt{MC}^{-1}(C)$ is constant on the two last rows.
   – the filter on the keys (from key schedule equations): a 120-bit filter as for the original attack against 23 rounds (15 equations on 8 bits each, see Table 3).
   – the filter on the keys (from equations describing the last round). Indeed, since $k_{in} \cup k_{out}$ generates the master key, K_{23} can be rewritten as $f(k_{in}) \oplus g(k_{out})$ where f and g are both linear and, because of the linearity of all the operations, the equation $C = \mathtt{MC}(\mathtt{SR}(\mathtt{SC}(S_{23}) \oplus K_{23}))$ can thus be rewritten as $C \oplus \mathtt{MC}(\mathtt{SR}(f(k_{in})) = \mathtt{MC}(\mathtt{SR}(\mathtt{SC}(S_{23}) \oplus g(k_{out})))$. This represents a 64-bit filter.
7. Repeat from Step 2 until the right key is retrieved.

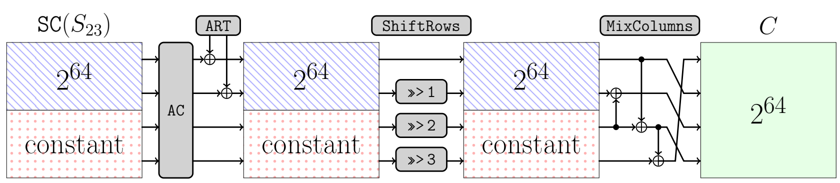


Fig. 8. Last round of the attack against 24 rounds.

The attack has to be repeated enough times so that the structure contains at least one pair following the differential. Since during Step 2 we generate 2^{64} pairs and since the probability of the differential is $2^{-105.9}$, the procedure has to be repeated $2^{41.9}$ times. Thus the data complexity is 2^{128} (i.e. the whole codebook is needed), the memory complexity is around 2^{314} 128-bit words and the time complexity $2^{256+64+41.9} = 2^{361.9}$ encryptions.

Note that previous improvements regarding both the memory and data complexities still apply and thus our attack has complexity: $\mathcal{D} = 2^{128-11} = 2^{117}$

plaintext/ciphertext pairs, $\mathcal{M} = 2^{194-11} = 2^{183}$ 128-bit words and $\mathcal{T} = 2^{361.9}$ encryptions.

5.3 An attack against 25 rounds of SKINNY-128-384

To mount a differential meet-in-the-middle attack against 25 rounds of SKINNY, we used the differential depicted in Figure 9. The best instantiation of this truncated differential has a probability of 2^{-131}. By fixing the difference of the active bytes to 0x32 at the input and to 0x64 at the output, we found several instantiations with a high enough probability: 2048 with probability 2^{-131}, 10240 with 2^{-132}, 28672 with 2^{-133} and finally 73728 trails with probability 2^{-134}. Thus, the probability of the depicted differential is higher than $2^{-116.5}$. This differential is then extended by 4 rounds to the plaintext and 5 rounds to the ciphertext to reach 24 rounds as depicted in Figure 10.

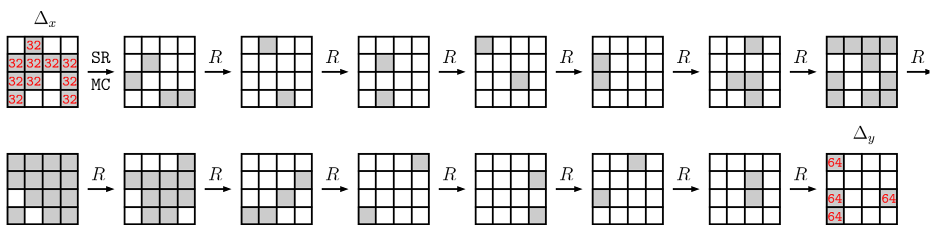


Fig. 9. Truncated differential trail for the attack on 25 rounds.

The key bytes involved in the attack are given in Table 4, leading to a complexity of $\mathcal{D} = 2^{128}$ data, $\mathcal{T} = 2^{256} \times 2^{116.5} = 2^{372.5}$ encryptions and $\mathcal{M} = 2^{249.5}$ 128-bit words. Furthermore, the dimension of the intersection between both k_{in} and k_{out} is once again $248+256-384 = 120$, which allows to reduce the memory complexity to $2^{129.5}$ 128-bit words.

As for the 23-round attack described above, this attack can be extended by one round without increasing its overall complexity. As a consequence, and after applying the data/time/memory trade-off presented Section 2.3, the complexity of our attack against 25 rounds is $\mathcal{D} = 2^{128-x}$ plaintext/ciphertext pairs, $\mathcal{M} = 2^{194-x}$ 128-bit words and $\mathcal{T} = 2^{372.5+x}$ encryptions. In this case, x cannot be higher than $(128-116.5)/2 = 5.75$. Furthermore, we can again apply this trade-off without increasing the time complexity and thus the final complexity is $D = 2^{122.3}$, $T = 2^{372.5}$ and $M = 2^{188.3}$.

6 Conclusion and open problems

We introduced in this work a new cryptanalysis technique, that we called the differential-MITM attack. We managed to successfully apply this new technique to SKINNY-128-384. Our attack against this variant of the SKINNY family of

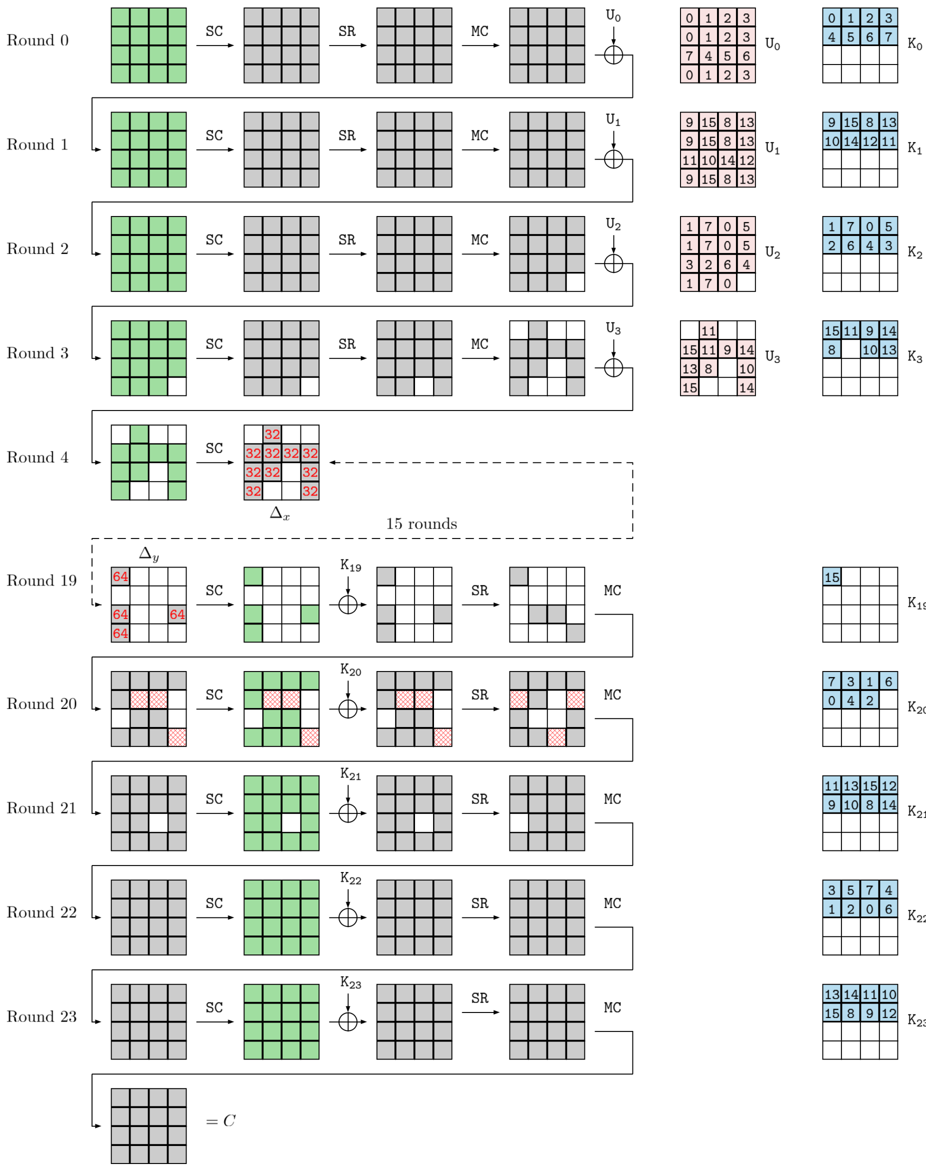


Fig. 10. Core attack against 24 rounds of `SKINNY-128-384`. Knowledge of blue key bytes allows to compute values of green ones and thus to propagate differences. No difference in both white and red bytes, but the red ones are required to compute green bytes. Indexes in round-key bytes are the indexes of the corresponding master key bytes.

Table 4. Round-key bytes involved in the 24-round core attack.

Byte	k_{in}	k_{out}	# equations
0	$K_0[0]$, $K_2[2]$	$K_{20}[4]$, $K_{22}[6]$	1
1	$K_0[1]$, $K_2[0]$	$K_{20}[2]$, $K_{22}[4]$	1
2	$K_0[2]$, $K_2[4]$	$K_{20}[6]$, $K_{22}[5]$	1
3	$K_0[3]$, $K_2[7]$	$K_{20}[1]$, $K_{22}[0]$	1
4	$K_0[4]$, $K_2[6]$	$K_{20}[5]$, $K_{22}[3]$	1
5	$K_0[5]$, $K_2[3]$	$K_{22}[1]$, $K_{24}[0]$	1
6	$K_0[6]$, $K_2[5]$	$K_{20}[3]$, $K_{22}[7]$	1
7	$K_0[7]$, $K_2[1]$	$K_{20}[0]$, $K_{22}[2]$	1
8	$K_1[2]$, $K_3[4]$	$K_{21}[6]$, $K_{23}[5]$	1
9	$K_1[0]$, $K_3[2]$	$K_{21}[4]$, $K_{23}[6]$	1
10	$K_1[4]$, $K_3[6]$	$K_{21}[5]$, $K_{23}[3]$	1
11	$K_1[7]$, $K_3[1]$	$K_{21}[0]$, $K_{23}[2]$	1
12	$K_1[6]$	$K_{21}[3]$, $K_{23}[7]$	0
13	$K_1[3]$, $K_3[7]$	$K_{21}[1]$, $K_{23}[0]$	1
14	$K_1[5]$, $K_3[3]$	$K_{21}[7]$, $K_{23}[1]$	1
15	$K_1[1]$, $K_3[0]$	$K_{19}[0]$, $K_{21}[2]$, $K_{23}[4]$	2

ciphers allowed us to provide the best single key attack against this construction, by reaching two more rounds than the previously best known attack.

The introduction of this new technique releases naturally numerous questions and opens many new research directions. First, we would like to further understand the link between the new attack and classical MITM attacks and how these two attacks can be compared. For example, we would like to identify for what kind of primitives the quantity of the involved key material in the differential MITM attack would be typically smaller compared to a classical MITM attack applied to the same cipher in a similar setting. We did some preliminary experiments on different ciphers by mounting both types of attacks in a comparable way and in some cases the amount of key bits to be guessed was smaller for the new attack while in some other cases this same amount was smaller for a classical MITM attack. It would be therefore interesting to be able to predict in an easy way, how this quantity compares for the two attacks.

As MITM attacks combine particularly well with the technique of bicliques, another natural question is whether differential MITM attacks combine well with bicliques as well. Furthermore, is it possible to find any concrete application where the combination of a differential MITM with bicliques could improve previous MITM or other attacks? Finally, can the technique of bicliques be combined with the method of partitions we proposed in the case of partial subkey additions, and how do they compare?

A last open question is whether instead of combining MITM techniques with differential attacks, one could successfully combine MITM with some other well-known family of cryptanalysis, such as for example linear or differential-linear attacks.

Acknowledgements. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 - acronym QUASYModo). It was also partially supported by the French Agence Nationale de la Recherche through the SWAP project under Contract ANR-21-CE39-0012 and through the DeCrypt project under Contract ANR-18-CE39-0007. Finally, the authors would like to thank the Dagstuhl Seminar 22141 on Symmetric Cryptography that gave the opportunity to the authors to advance this collaboration.

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. Lecture Notes in Computer Science, vol. 5381, pp. 103–119. Springer (2008)
2. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 70–89. Springer (2009)
3. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 321–345. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_16, https://doi.org/10.1007/978-3-319-66787-4_16
4. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
5. Biham, E., Biryukov, A., Shamir, A.: Miss in the middle attacks on IDEA and Khufu. In: Knudsen, L.R. (ed.) FSE '99. Lecture Notes in Computer Science, vol. 1636, pp. 124–138. Springer (1999)
6. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack - rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 340–357. Springer (2001)
7. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO '90. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990)
8. Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Brickell, E.F. (ed.) CRYPTO '92. Lecture Notes in Computer Science, vol. 740, pp. 487–496. Springer (1992)
9. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 344–371. Springer (2011)
10. Bordes, N., Daemen, J., Kuijsters, D., Assche, G.V.: Thinking outside the superbox. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 337–367. Springer (2021). https://doi.org/10.1007/978-3-030-84252-9_12, https://doi.org/10.1007/978-3-030-84252-9_12

11. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 222–240. Springer (2013)
12. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks (full version). IACR Cryptol. ePrint Arch. p. 324 (2013), http://eprint.iacr.org/2013/324
13. Choudhuri, A.R., Maitra, S.: Differential cryptanalysis of Salsa and ChaCha – An evaluation with a hybrid model. Cryptology ePrint Archive, Paper 2016/377 (2016), https://eprint.iacr.org/2016/377, https://eprint.iacr.org/2016/377
14. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE '97. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)
15. Delaune, S., Derbez, P., Huynh, P., Minier, M., Mollimard, V., Prud'homme, C.: Efficient methods to search for best differential characteristics on SKINNY. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 2021, Part II. Lecture Notes in Computer Science, vol. 12727, pp. 184–207. Springer (2021)
16. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. Lecture Notes in Computer Science, vol. 5086, pp. 116–126. Springer (2008)
17. Diffie, W., Hellman, M.: Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. Computer **10**(6), 74–84 (1977)
18. Dong, X., Hua, J., Sun, S., Li, Z., Wang, X., Hu, L.: Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 278–308. Springer (2021)
19. Dunkelman, O., Keller, N., Shamir, A.: Improved single-key attacks on 8-round AES-192 and AES-256. In: Abe, M. (ed.) ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 158–176. Springer (2010)
20. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. Lecture Notes in Computer Science, vol. 4859, pp. 86–100. Springer (2007)
21. Feistel, H.: Cryptography and computer privacy. Scientific american **228**(5), 15–23 (1973)
22. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 56–75. Springer (2010)
23. Hadipour, H., Sadeghi, S., Eichlseder, M.: Finding the impossible: Automated search for full impossible differential, zero-correlation, and integral attacks (preliminary version). IACR Cryptol. ePrint Arch. p. 1147 (2022), https://eprint.iacr.org/2022/1147
24. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. Lecture Notes in Computer Science, vol. 7707, pp. 202–221. Springer (2012)
25. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. Lecture Notes in Computer Science, vol. 8874, pp. 274–288. Springer (2014)
26. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of nlfsr-based cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 130–145. Springer (2010)

27. Knudsen, L.: DEAL-a 128-bit block cipher. complexity **258**(2), 216 (1998)
28. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE '94. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994)
29. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) CRYPTO '94. Lecture Notes in Computer Science, vol. 839, pp. 17–25. Springer (1994)
30. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINTcipher: The invariant subspace attack. In: Rogaway, P. (ed.) CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 206–221. Springer (2011)
31. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT '93. Lecture Notes in Computer Science, vol. 765, pp. 386–397. Springer (1993)
32. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In: Dunkelman, O. (ed.) Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5665, pp. 260–276. Springer (2009)
33. Naya-Plasencia, M.: How to improve rebound attacks. In: Rogaway, P. (ed.) CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 188–205. Springer (2011)
34. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016), http://www.choco-solver.org
35. Shi, D., Sun, S., Derbez, P., Todo, Y., Sun, B., Hu, L.: Programming the demirci-selçuk meet-in-the-middle attack with constraints. In: Peyrin, T., Galbraith, S.D. (eds.) ASIACRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 3–34. Springer (2018)
36. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)
37. Tolba, M., Abdelkhalek, A., Youssef, A.M.: Impossible differential cryptanalysis of reduced-round SKINNY. In: Joye, M., Nitaj, A. (eds.) AFRICACRYPT 2017. Lecture Notes in Computer Science, vol. 10239, pp. 117–134 (2017)
38. Wagner, D.A.: The boomerang attack. In: Knudsen, L.R. (ed.) FSE '99. Lecture Notes in Computer Science, vol. 1636, pp. 156–170. Springer (1999)
39. Yang, D., Qi, W., Chen, H.: Impossible differential attacks on the SKINNY family of block ciphers. IET Inf. Secur. **11**(6), 377–385 (2017)

A The 8-bit S-box used in `SKINNY-128-t`

The 8-bit S-box `S` used for the 128-bit variants of `SKINNY` is given below. All values are in hexadecimal notation.

```
S = [ 65 4c 6a 42 4b 63 43 6b 55 75 5a 7a 53 73 5b 7b
      35 8c 3a 81 89 33 80 3b 95 25 98 2a 90 23 99 2b
      e5 cc e8 c1 c9 e0 c0 e9 d5 f5 d8 f8 d0 f0 d9 f9
      a5 1c a8 12 1b a0 13 a9 05 b5 0a b8 03 b0 0b b9
      32 88 3c 85 8d 34 84 3d 91 22 9c 2c 94 24 9d 2d
      62 4a 6c 45 4d 64 44 6d 52 72 5c 7c 54 74 5d 7d
      a1 1a ac 15 1d a4 14 ad 02 b1 0c bc 04 b4 0d bd
      e1 c8 ec c5 cd e4 c4 ed d1 f1 dc fc d4 f4 dd fd
      36 8e 38 82 8b 30 83 39 96 26 9a 28 93 20 9b 29
      66 4e 68 41 49 60 40 69 56 76 58 78 50 70 59 79
      a6 1e aa 11 19 a3 10 ab 06 b6 08 ba 00 b3 09 bb
      e6 ce ea c2 cb e3 c3 eb d6 f6 da fa d3 f3 db fb
      31 8a 3e 86 8f 37 87 3f 92 21 9e 2e 97 27 9f 2f
      61 48 6e 46 4f 67 47 6f 51 71 5e 7e 57 77 5f 7f
      a2 18 ae 16 1f a7 17 af 01 b2 0e be 07 b7 0f bf
      e2 ca ee c6 cf e7 c7 ef d2 f2 de fe d7 f7 df ff ]
```

B Enumeration Procedure of k_{out} for the 23-round Attack

In this section we describe a procedure to retrieve the possible values of k_{out} in the case where the pair of ciphertexts is given. Its complexity is 2^{128} simple operations, showing that the data/time/memory trade-off described in Section 2.3 can be applied without increasing the time complexity. Since the key is not applied on the full state, the procedure is more complex than a classical rebound but still relies on the fact that, on average, knowing the differences at both the input and output of an Sbox leads to one pair of actual values.

The steps of our enumeration procedure are depicted on Figure 11. The main idea is to propagate differences from the ciphertexts to Round 18, get the actual values since differences in this internal state are fully known, and propagate them back to the ciphertexts to obtain the key material we want. Except at Step 1 in which we guess 8 values, all next steps perform one guess each. Steps 7 and 8 both have a probability of success of 2^{-8} and thus we expect only $2^{13\times 8} = 2^{104}$ partial solutions at the end of Step 8.

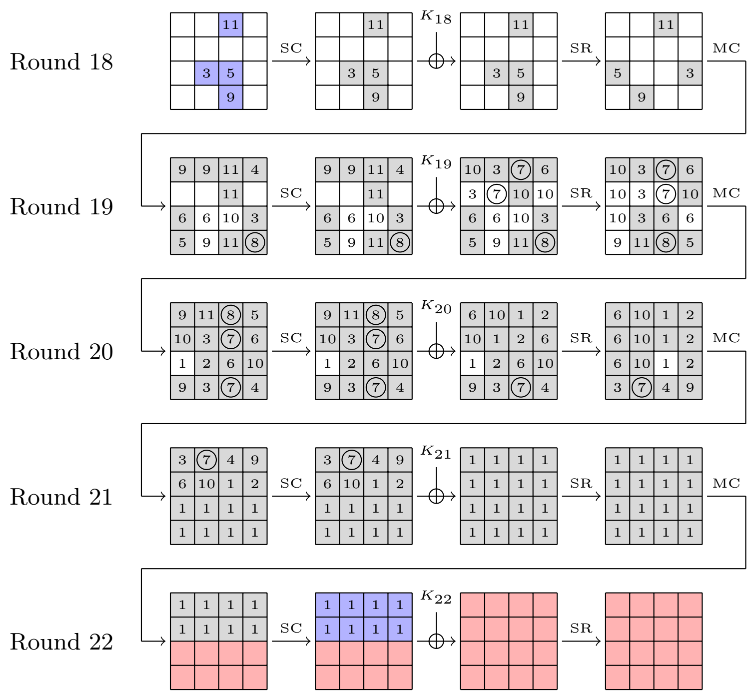


Fig. 11. Enumeration procedure of k_{out} for the 23-round attack. Differences in blue cells and actual values in red cells are known. No difference in white cells.