Cryptanalysis of SKINNY in the Framework of the SKINNY 2018-2019 Cryptanalysis Competition

Patrick Derbez, Virginie Lallemand, Aleksei Udovenko

Cryptanalysis of SKINNY in the Framework of the SKINNY 2018-2019 Cryptanalysis Competition

Patrick Derbez[1⋆], Virginie Lallemand[2,3], and Aleksei Udovenko[4]

[1] Univ Rennes, CNRS, IRISA, F-35000 Rennes, France
patrick.derbez@irisa.fr
[2] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[3] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany
virginie.lallemand@loria.fr
[4] SnT and CSC, University of Luxembourg, Luxembourg
aleksei.udovenko@uni.lu

Abstract. In April 2018, Beierle et al. launched the 3rd SKINNY cryptanalysis competition, a contest that aimed at motivating the analysis of their recent tweakable block cipher SKINNY. In contrary to the previous editions, the focus was made on practical attacks: contestants were asked to recover a 128-bit secret key from a given set of 2^{20} plaintext blocks. The suggested SKINNY instances are 4- to 20-round reduced variants of SKINNY-64-128 and SKINNY-128-128. In this paper, we explain how to solve the challenges for 10-round SKINNY-128-128 and for 12-round SKINNY-64-128 in time equivalent to roughly 2^{52} simple operations. Both techniques benefit from the highly biased sets of messages that are provided and that actually correspond to the encryption of various books in ECB mode.

Keywords: Cryptanalysis, SKINNY, Low Data attack, Truncated Differential, Higher Order Differential, Integral Cryptanalysis

1 Introduction

In order to motivate external cryptanalyses of their family of ciphers, SKINNY designers launched several one-year competitions. The first one started in 2016 and called for cryptanalyses of small-scaled variants of 18 up to 26 rounds of SKINNY-64-128, and of 22 up to 30 rounds of SKINNY-128-128. The two papers that won the competition are [1] for being the first submission that attacks up to 20 rounds of SKINNY-64-128 and [9] for being the first submitted work to successfully attack up to 23 rounds of SKINNY-64-128.

⋆ Patrick Derbez was supported by the French Agence Nationale de la Recherche through the CryptAudit project under Contract ANR-17-CE39-0003. This work was initiated while Virginie Lallemand was with the Horst Görtz Institute for IT Security at the Ruhr-Universität Bochum. The work of Aleksei Udovenko was partially supported by the Fonds National de la Recherche, Luxembourg (project reference 9037104).

The challenges launched in 2017 were similar, except that the number of rounds one has to break were higher. Nobody won these contests.

The last competition started on the 1st of April 2018 and ended on February 28, 2019. This time, the goal was to mount a practical key recovery attack of small-scaled versions of SKINNY for which sets of only 2^{20} pairs (plaintext, ciphertext) were provided. The designers offered rewards for the teams that would break the maximum number of rounds for SKINNY-64-128 or SKINNY-128-128[5].

Contributions. In this paper, we describe our practical attacks on 12-round SKINNY-64-128 and on 10-round SKINNY-128-128 in the setting of the third SKINNY Cryptanalysis Competition. At the time of writing and as far as we know, these attacks are the ones that cover the largest number of rounds.

Table 1. Complexities of our attacks: the data complexity corresponds to the number of messages that are actually exploited in the attack, while time complexity is expressed in number of basic operations.

Version	Rounds	Technique	Data	Time	Memory
SKINNY-64-128	12	Truncated diff.	64	$2^{51.95}$	256 GB
SKINNY-128-128	10	2nd-order truncated diff.	24	2^{52}	0.5 GB

Paper Organization. After briefly recalling the description of the SKINNY block cipher (Section 2), we study the structure of the provided set of 2^{20} messages in Section 3 and show that the observed bias comes from the fact that the plaintexts are sentences of various famous books encrypted in ECB mode. We then describe in Section 4 how we can benefit from this to find distinguishers that would not have been possible if the input messages were uniformly distributed. More into details, Section 5 reports a truncated differential attack that works with time close to 2^{52} operations. In this attack, a careful study of the tweakey schedule showed that a 4-bit guess can be saved, while the key recovery algorithm is optimised so that the required memory stays practical. In Section 6, we use second-order truncated differentials to break 10-round SKINNY-128-128 with 2^{52} basic operations.

The source codes of our attacks are available at

http://skinnysac19.gforge.inria.fr/

[5] All the information on the competitions and on the cipher in general can be found on https://sites.google.com/site/skinnycipher/home.

2 A brief description of SKINNY

SKINNY is a family of lightweight tweakable block ciphers that was designed by Beierle et al. and published at Crypto 2016 [3]. The objective pursued (and achieved) by the designers was to propose a family of ciphers that would have similar performances to those of the NSA cipher Simon [2] but with strong security arguments on top.

The SKINNY ciphers follow the TWEAKEY framework from [6] so each instance takes as input a plaintext P and a tweakey TK to produce a ciphertext C. The different variants in the family are noted SKINNY-n-t, where n represents the block size (64 or 128 bits) and where t is the tweakey size (n, $2n$ or $3n$).

As for the AES and many other SPN, the internal state is organised in a matrix of 4×4 elements. Depending on the block size, each cell is either a nibble (4 bits, when $n = 64$) or a byte (8 bits, when $n = 128$). Similarly, the tweakey is arranged in $z = t/n$ such 4×4 matrices, noted TK1 up to TKz. Both the messages and the tweakey states are loaded row-wise, and we number the nibbles in this way (see Figure 8).

In the following we only provide a high-level description of SKINNY and refer to the original specification [3] for further details.

Round function. SKINNY round function is illustrated in Figure 1. It can be noted that contrary to most ciphers, the (tweak)key is added after the non-linear operation (SubCells), and not before.

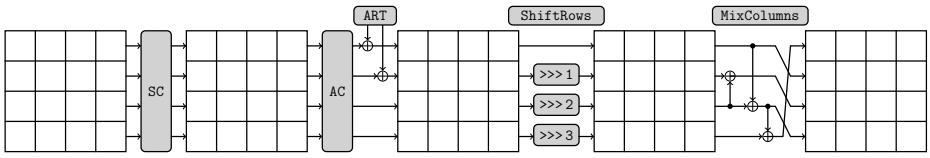

Fig. 1. SKINNY round function (figure credits: [5]).

Also, no whitening keys are used and the last MixColumns is not omitted, which means that all the rounds iterate exactly the same sequence of operations, that are:

- SubCells (SC): Each of the 16 cells of the internal state is modified by the 4×4 Sbox $\mathcal{S}_4$ described in Table 4 if each cell is a nibble, or by the 8×8 Sbox $\mathcal{S}_8$ described in Table 5 if each cell is a byte.
- AddConstants (AC): Constants generated by an LFSR are added in the first 3 cells of the first column.
- AddRoundTweakey (ART): The tweakey material is XORed to the first two lines of the state.

– `ShiftRows` (`SR`): The second, third and fourth rows of the internal state are respectively right-rotated by 1 cell, 2 cells, and 3 cells.
– `MixColumns` (`MC`): Each input column of the internal state is multiplied by the following binary matrix M:

$$M = \begin{pmatrix} 1\,0\,1\,1 \\ 1\,0\,0\,0 \\ 0\,1\,1\,0 \\ 1\,0\,1\,0 \end{pmatrix}$$

The number of times the round function is repeated for each version of `SKINNY` is detailed in Table 2.

Table 2. Number of iterated rounds for each version of the cipher, depending on the tweakey size ($t = z \times n$) and on the block size n.

	Tweakey size		
block size	$z = 1$	$z = 2$	$z = 3$
$n = 64$	32	36	40
$n = 128$	40	48	56

Tweakey Schedule. `SKINNY` follows the Tweakey framework defined in [6] that allows an user to add an additional parameter, the so-called tweak. In the setting of the third cryptanalysis competition, the 128 bits of tweakey of `SKINNY`-64-128 or `SKINNY`-128-128 are unknown, and consequently the 128 bits are considered as key only.

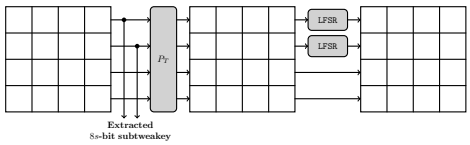

Fig. 2. Tweakey schedule of `SKINNY` (figure credits: [5])

The tweakey schedule is represented in Figure 2. For `SKINNY`-128-128 there is only one tweakey block (TK1) organised in a square matrix of bytes. At each round, the first two lines of the matrix are extracted and used as round tweakey in the step `ART` of the round function. Then, the bytes are permuted by the permutation P_T, given by:

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

This process is repeated for the following rounds.

In the case of SKINNY-64-128 we have $z = 2$, which means that we have two tweakey states (TK_1 and TK_2) organised as 4×4 matrices of nibbles. A round tweakey is made by extracting the first two rows of each tweakey state and XORing them together. To produce the next tweakey, TK_1 and TK_2 are first updated by applying the permutation P_T that reorder their 16 nibbles. The 8 nibbles of the first two rows of TK_2 are then further modified by the following linear operation:

$$L : (x_3||x_2||x_1||x_0) \mapsto (x_2||x_1||x_0||x_3 \oplus x_2)$$

where x_0 is the LSB of the cell.

3 Remark on the Provided Messages

While looking for messages with specific patterns, we realized that the plaintexts provided for the challenges were not uniformly distributed.

To illustrate this, we provide in Figure 3 the distribution of the value of nibble 0 (top left corner in the Skinny internal state) and of nibble 15 (bottom right) in the set provided for the 12-round attack on SKINNY-64-128.

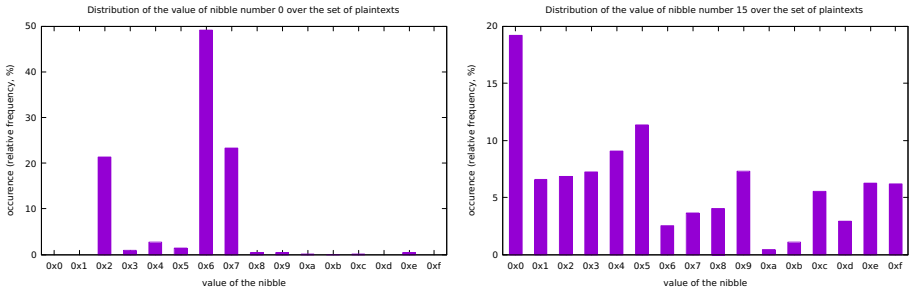

Fig. 3. Distribution of the value of nibble 0 (left) and of nibble 15 (right) of the 2^{20} plaintexts provided for the 12-round attack on SKINNY-64-128.

In fact, the bias observed in Figure 3 is present on the other nibble positions too, and we made the following observations:

- All the nibbles positioned at even indices (when considering the numbering of Figure 8) have a distribution similar to the one of the left bar chart of Figure 3: the most frequent value is 0x6 (occurring roughly half of the time), followed by 0x7 and 0x2. The other values are very rare.
- The nibbles at odd positions don't have such a strong bias. Still, some values are more frequent than the others, like 0x0 that appears in one case out of 5.

It is rather direct to make the link between this distribution and the one of a text in UTF-8 code: indeed, the first hint comes from the fact that the UTF-8 code of the lower-case letters goes from 0x61 to 0x7a, which explains the overwhelming occurrence of the nibble 0x6 (followed by the nibble 0x7) in the distribution of nibbles at even positions. Also, a character that is ought to appear frequently is the space, encoded by 0x20. This one explains the third dominant higher nibble value (0x2) and the high number of occurrences of 0x0 in the lower nibbles.

This guess was confirmed once we printed the plaintexts. For instance, looking at the messages given for the challenge on 4 rounds SKINNY-64-128, we read:

> Project Gutenberg's Alice's Adventures in Wonderland, by Lewis Carroll
> This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever.

And few lines later, confirming that this is the book, one can read:

> [...] when suddenly a White Rabbit with pink eyes ran close by her. There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!'

Other data sets correspond to other books (for instance Metamorphosis, by Franz Kafka or The Prince, by Nicolo Machiavelli).

This high bias in the messages implies that we have some collisions on the plaintext values. In the file provided for 12 rounds of SKINNY-64-128 we counted 925615 different pairs ($2^{19.82}$, so $2^{16.90}$ collisions) out of the 2^{20} provided. The plaintext that appears the most corresponds to $\boxed{.\text{"}\ \text{“}}$ (dot (0x2e), closed quote (0xe2809d), space (0x20), open quote (0xe2809c)) with 289 occurrences.

In the following, we detail how these biases work to our advantage by inducing a set of messages with low entropy, helping us pushing further our attacks.

4 Mining Truncated Differentials in the Provided Datasets

Any human-readable text has a low entropy and so does the book whose encryption was provided in the SKINNY 2018-2019 cryptanalysis competition. This fact can be exploited by a cryptanalyst, who can devise attacks that would be ineffective in the classic known-plaintext scenario with uniformly distributed plaintexts. In this section, we show that many pairs or even quadruples of plaintext blocks can be found in the provided datasets for which particular (differential) zero-sum properties hold with probability 1 after 6 rounds of SKINNY-64-128 and 7 rounds of SKINNY-128-128.

We restrict our search to truncated differentials holding with probability 1. Such distinguishers require only a few message pairs fitting them to provide a strong filtering procedure for the key recovery. In addition to this, we also

consider second-order differentials [8,7]: instead of studying the propagation of differences between 2 messages as it is done with (first-order) differentials, we look at the differences between a set of 2^2 messages. Note that finding such sets of four distinct plaintext blocks that sum to zero has higher chances to happen when these blocks differ only in a few cells. In our case, the low entropy of the text facilitates such events.

4.1 Search Strategy

Our general search strategy is as follows. Instead of enumerating all possible truncated differentials in the structure, we search precisely for those differentials that fit the given set of plaintexts for any possible key. In order to find them, we encrypt the plaintexts under several randomly chosen keys and search for differential properties in the obtained ciphertexts. This empirical approach allows tailoring the search to the given dataset.

Let P denote the set of provided plaintexts. We fix a `SKINNY` instance, a number of rounds r in the target distinguisher, and a maximum number m ($1 \leq m \leq 15$) of active cells after the first round.

First, we compute all subsets of plaintexts such that all plaintexts inside any such subset take the same value in at least $16 - m$ cells, i.e. at most m cells are active. In this step we exploit the fact that the first tweakey addition is done after the non-linear operation and thus differences after the first round do not depend on the key. In order to compute these subsets, we enumerate all possible patterns of active cells and for each pattern we simply group the plaintexts by the values of inactive cells.

We further process each such subset $S \subseteq P$ as follows. We choose t random keys $k_0, \ldots, k_{t-1}$ and encrypt each plaintext $p \in S$ using r-round `SKINNY` with those keys. For each cell i of the output state, we record the vector $v_{p,i}$ of values computed in that cell:

$$v_{p,i} = \Big((E_{k_0}(p))_i, (E_{k_1}(p))_i, \ldots, (E_{k_{t-1}}(p))_i \Big),$$

where E_k denotes r-round `SKINNY` encryption with the key k. Such vectors are $4t$ bits long in the case of `SKINNY`-64 and $8t$ bits long in the case of `SKINNY`-128.

Assume that we find a 2-vector collision, i.e. $v_{p,i} = v_{p',i'}$ for some $(p, i) \neq (p', i')$. It is easy to see that $i \neq i'$ is unlikely because of the use of random keys. Therefore, we obtain $p \neq p'$ and $i = i'$ and we conclude that the chosen `SKINNY` instance with overwhelming probability has zero difference in cell i after r rounds of encryption of plaintexts p and p'. By increasing t, the probability of error can be made negligible. Furthermore, any such concrete truncated differential is usually easy to verify.

If no 2-vector zero-sum occurred, we aim to find 4-vector zero-sums, which would possibly lead to distinguishers on more rounds. This can be done in time $O(|S|^2 t)$ for each subset $S \subseteq P$ by computing all pairwise vector differences and finding a collision. More precisely, for each pair of elements $v_{p,i}, v_{p',i'} \in S$ we

add their difference $v_{p,i} \oplus v_{p',i'}$ to a hash table. A collision of such differences means that we have found four vectors $v_{p,i}, v_{p',i'}, v_{p'',i''}, v_{p''',i'''}$ such that

$$v_{p,i} \oplus v_{p',i'} \oplus v_{p'',i''} \oplus v_{p''',i'''} = 0.$$

The converse is also true, i.e., any 4-vector zero-sum implies a difference collision that this approach would find.

In practice, there are two cases that may happen:

1. the four vectors are $v_{p,i}, v_{p',i}, v_{p,i'}, v_{p',i'}$ with $p \neq p', i \neq i'$ (i.e. $p = p'', p' = p''', i = i''', i' = i''$);
2. the four vectors are $v_{p,i}, v_{p',i}, v_{p'',i}, v_{p''',i}$ with p, p', p'', p''' pairwise distinct (i.e. $i = i' = i'' = i'''$).

The first case corresponds to a truncated differential with input difference $\Delta p = p \oplus p'$ and the property that $\Delta c_i = \Delta c_{i'}$, where $\Delta c = c \oplus c'$, c and c' are ciphertexts corresponding to r-round SKINNY encryptions of plaintexts p and p'. Such event may occur when the input difference of MixColumns has inactive cells. For example,

$$\texttt{MixColumn}(\delta, 0, 0, 0) = (0, \delta, 0, \delta)$$

for any cell difference δ.

The second case corresponds to the zero-sum property

$$c_i \oplus c'_i \oplus c''_i \oplus c'''_i = 0,$$

where c, c', c'', c''' are ciphertexts corresponding to r-round SKINNY encryptions of plaintexts p, p', p'', p'''. This event usually requires that the plaintexts form a second-order difference, i.e. they also sum to zero.

4.2 Truncated Distinguishers of SKINNY Instances

We applied this search strategy for the challenge datasets of 12-round SKINNY-64-128 and 10-round SKINNY-128-128 with at most $m = 3$ active bytes after the first round. The results are summarized in Table 3.

For 12-round SKINNY-64-128, we found distinguishers for up to 7 rounds and only of the first type. For example, there are 57 pairs of plaintexts in the dataset for which $\Delta c_0 = \Delta c_{12}$ after 7 rounds. An example of such plaintext pair (p, p') is given by

$$p = \boxed{\texttt{a pouch,}},$$
$$p' = \boxed{\texttt{a pause,}}.$$

We use this distinguisher to attack 12-round SKINNY-64-128 and recover the secret key. The process is described in Section 5.

For the 10-round SKINNY-128-128, we found distinguishers for up to 6 rounds among which both distinguisher types are present. For example, there are 9

Table 3. Truncated distinguishers with maximum number of rounds found using our approach in the datasets of 12-round SKINNY-64-128 and 10-round SKINNY-128-128. Bold font shows distinguishers used in our attacks.

Version	Rounds/ Dataset Rounds	Property	#Plaintexts in Structure	#Structures in Dataset
SKINNY-64-128	7/12	$\Delta c_2 = \Delta c_{14}$	2	76
		$\Delta c_5 = \Delta c_{13}$	2	76
		$\mathbf{\Delta c_0 = \Delta c_{12}}$	**2**	**57**
		$\Delta c_7 = \Delta c_{15}$	2	57
		$\Delta c_3 = \Delta c_{15}$	2	50
		$\Delta c_6 = \Delta c_{14}$	2	50
		$\Delta c_1 = \Delta c_{13}$	2	22
		$\Delta c_4 = \Delta c_{12}$	2	22
SKINNY-128-128	6/10	$\Delta c_3 = \Delta c_{15}$	2	12180
		$\Delta c_6 = \Delta c_{14}$	2	12093
		$\Delta c_5 = \Delta c_{13}$	2	2443
		$\Delta c_2 = \Delta c_{14}$	2	2412
		$\Delta c_4 = \Delta c_{12}$	2	600
		$\Delta c_1 = \Delta c_{13}$	2	598
		$\Delta c_7 = \Delta c_{15}$	2	430
		$\Delta c_0 = \Delta c_{12}$	2	413
		$\Delta c_3 = \Delta c_7$	2	16
		$\Delta c_1 = \Delta c_5$	2	11
		$\Delta c_0 = \Delta c_4$	2	8
		$\Delta c_2 = \Delta c_6$	2	1
		$\bigoplus \mathbf{c_9 = 0}$	**4**	**9**
		$\bigoplus c_{10} = 0$	4	7
		$\bigoplus c_8 = 0$	4	5
		$\bigoplus c_{11} = 0$	4	1

quadruples of plaintexts for which after 6 rounds of SKINNY-128-128 the cell number 9 sums to zero. An example of such quadruple is given by

$$p = \boxed{\texttt{for a moment an}},$$
$$p' = \boxed{\texttt{for a moment at}},$$
$$p'' = \boxed{\texttt{for a moment in}},$$
$$p''' = \boxed{\texttt{for a moment it}}.$$

We use this distinguisher to recover the secret key of this SKINNY instance in Section 6.

Figure 4 depicts one representative of each of the properties given in Table 3. As can be seen in the table, each distinguisher has 3 other variants, corresponding to its column rotations.

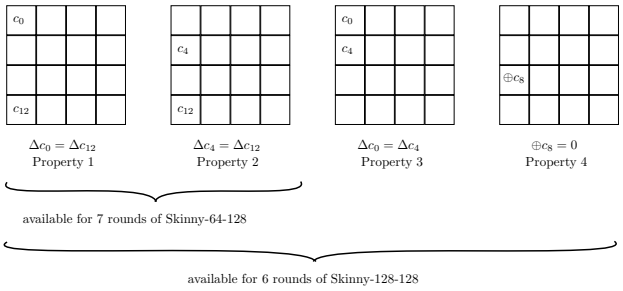

Fig. 4. Four properties found with our search strategy

5 Attacking 12-round SKINNY-64-128

5.1 Our Truncated Differential Trail

To attack 12 rounds, we use the truncated differential distinguisher described in Section 4.2 which benefits from the not-so-uniform distribution of the provided messages. As depicted in Figure 5, we consider a truncated differential path that starts at round 2 with a difference of only one nibble, in $X_1[12]$. After 6 rounds, the first and last nibbles of the first column, denoted by "U_1" and "U_2" in the figure, are always equal.

As already explained in Section 4, the fact that the first tweakey addition is made after the non-linear layer makes the search for conforming pairs straightforward. For any pair, an attacker can compute the difference at the end of round 1 to check if only $X_1[12]$ is active. If this condition is met, the rest of the characteristic (up to X_7) is fulfilled with probability 1.

Fig. 5. Truncated differential trail used in our attack of SKINNY-64-128. The white cells are inactive, the green ones are active for sure, while the yellow ones might be active. If one pair of messages only differs on $X_1[12]$, the difference in the two nibbles $X_7[0]$ and $X_7[12]$ (denoted U_1 and U_2) are equal with probability one.

Once the attacker has determined the conforming pairs, she can deduce information on the key by simply making guesses on the last round keys and checking if she obtains equality between the differences in the nibbles $X_7[0]$ and $X_7[12]$.

5.2 From the Truncated Differential Path to the Attack: the Key Recovery Part

The first step of our attack consists in exploring the set of 2^{20} messages to extract the pairs that comply with the start of the truncated differential path (that are pairs with a single active nibble at the beginning of round 1, at $X_1[12]$). There are 57 such pairs[6], which contrasts with what one would have obtain for a uniformly distributed set (since the occurrence probability would have been 2^{-60}).

Once this is done, an attacker makes key guesses to compute the differences in $X_7[0]$ and $X_7[12]$ to check if they are equal. If they are not, the guessed key is for sure incorrect, while if the equality is verified for all the considered pairs the key guess might be the right one.

The key and state nibbles that need to be guessed are represented in Figure 7.

Out of the 19 tweakey nibbles that need to be guessed, 1 can be saved thanks to the tweakey schedule relations:

Saving a nibble by taking into account the tweakey schedule. The definition of the tweakey schedule of SKINNY implies that the round tweakeys of all the even rounds (similarly of all the odd rounds) are related together by linear relations. In the case of SKINNY-64-128, this might lead to a reduction in the number of key guesses if we consider more than 4 consecutive rounds in the key recovery.

To see how we can benefit from this in our attack, let us denote by x_i $(0 \leq i \leq 15)$ the nibbles of the first tweakey state (TK_1) of tk_7 and by y_i $(0 \leq i \leq 15)$ the nibbles of its second tweakey state (TK_2) (see Figure 6).

With these notations, $tk_7[0]$, $tk_9[2]$ and $tk_{11}[4]$ can be written as follows:

$$\begin{aligned} tk_7[0] &= x_0 \oplus y_0 \\ tk_9[2] &= x_0 \oplus L(y_0) \\ tk_{11}[4] &= x_0 \oplus L(L(y_0)) \end{aligned}$$

where L denotes one application of the linear operation (that can be seen as a LFSR) defined in Section 2. If the attacker has already made a guess on the values of $tk_9[2]$ and $tk_{11}[4]$, the previous relations imply that she can deduce the value of $tk_7[0]$ by computing:

$$tk_7[0] = L^{-1}(tk_9[2] \oplus tk_{11}[4]) \oplus tk_9[2] \tag{1}$$

[6] As seen previously, another possibility would have been to use the 76 pairs available for the same trail shifted by 2 columns, that are starting with only one nibble active at position 14 after 1 rounds and resulting into nibbles in position 2 and 14 that are equal at the end of round 7 (see Table 3).

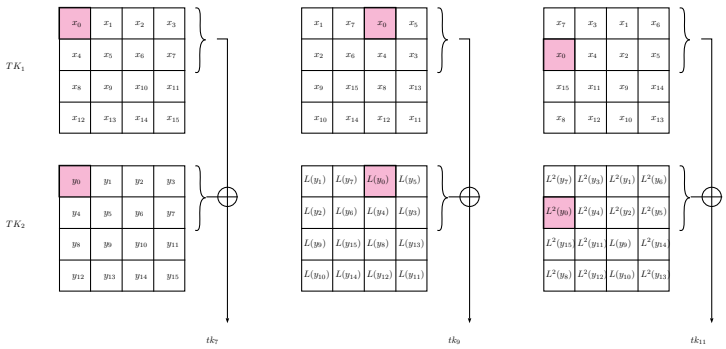

Fig. 6. Close up of the tweakey schedule in the final rounds.

Which induces a save of a 4-bit guess in the key recovery process.

Algorithmic Description of the Key Recovery. In addition to Figure 7, we give in Algorithm 2 in Appendix a step by step description of the naive way to implement the recovery of the round tweakeys involved in the computation of U_1 and U_2. Quite naturally, the key recovery is organised so that the key nibbles that are required to compute both U_1 and U_2 are guessed first. Then, one part of the algorithm is dedicated to the guess of the remaining key nibbles required to obtain U_1 and another one, independently, deals with U_2. Remember that the numbering of the state nibbles that is used is the one of Figure 8.

Note that we don't give the tedious details of how the nibbles are computed, as we consider that this is rather obvious from the cipher description. Indeed, as per usual in key recovery algorithms, the operations consist in decrypting partially from the ciphertext, nibble after nibble, for the positions that are required to get U_1 and U_2. Just to give an example, $X_{11}[7]$ is obtained by computing $X_{11}[7] = \mathcal{S}_4^{-1}(Z_{11}[7] \oplus tk_{11}[7])$.

In the naive version of Algorithm 2, T_1 (respectively T_2) contains 2^{44} lines composed of the 32 computed values of U_1 (resp. U_2) concatenated to the tweakey guess of 11 nibbles. To reduce this huge memory requirement we exploit further the fact that U_1 and U_2 both depend on $tk_{11}[7]$, $tk_{10}[6]$, $tk_{11}[3]$ and $tk_9[2]$.

The idea consists in going over all the possible values of the 4 common nibbles, and to repeat the building of the two tables once the values of these 4 are fixed (so without storing them in the tables). This optimisation (and others) are presented in Algorithm 1.

In this way, the two tables V_1 and V_2 contain 2^{28} rows. Each row is made of $4 \times 2^4 = 64$ bits (corresponding to the computed nibble of difference U_1 or U_2 for all the 2^4 input pairs). In this first step, we don't keep track of the value of the other tweakey nibbles in order to reduce memory as far as possible. The goal is only to get a reduced set of possibilities for the 4 nibbles $tk_{11}[7]$, $tk_{10}[6]$, $tk_{11}[3]$ and $tk_9[2]$. Once this is done, a second iteration gives the desired result.

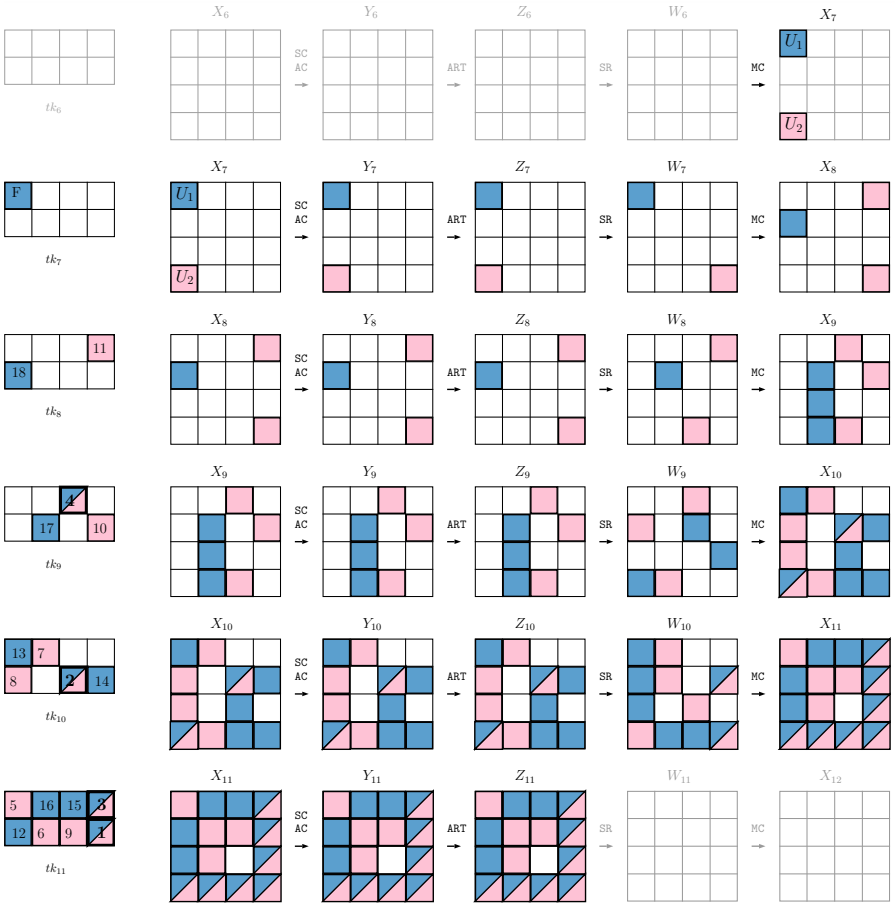

Fig. 7. Nibble dependencies in the key recovery process. We want to recover the difference U_1 and U_2. To compute U_1 from a pair of ciphertexts we need the value of the blue nibbles of the internal state and of the blue nibble of the round tweakeys. Similarly, the pink nibbles are required to recover the difference U_2. The numbers correspond to the order in which is done the key guessing: we start by the nibbles that are needed for both U_1 and U_2. The round tweakey nibble denoted by F comes for free if $tk_{11}[4]$ and $tk_9[2]$ are known, which is why we mark $tk_9[2]$ as required for both U_1 and U_2.

The second iteration takes as parameter the value of the previous match so that the filtering is quicker. Matching on these at the second iterations means we are considering the same value for the other nibbles of key guesses.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

Fig. 8. Numbering of the nibbles in the internal state.

Similarly to the naive algorithm, building each table requires to guess 2^{44} bits. Once V_1 and V_2 are filled, they are sorted[7]. We then merge the two tables on the 2^4 4-bit values of U_1 and U_2: the probability of a match is equal to 2^{-64}, which implies that $2^{16} \times 2^{28+28-64} = 2^{16} \times 2^{-8}$ values of the 4 tweakey nibbles $tk_{11}[7]$, $tk_{10}[6]$, $tk_{11}[3]$ and $tk_9[2]$ survive this filter.

To decide on which of these is correct, we repeat the previous process for the same 2^4 pairs plus 2^4 new ones for the 2^8 surviving values (this time we keep track of all the guessed values in tk_{11}). Only one value of tk_{11} passes the test.

In this optimised version, we focus on recovering tk_{11} only. To recover the other round tweakeys, the attacker peels off the last round by using tk_{11} and launches a similar and faster attack on 11-round `SKINNY`, and so on until the master key can be recovered.

Total Complexity. The complexity of the key recovery is dominated by the cost of recovering tk_{11}, that can be decomposed as follows:

$$2^{16} \times \Big(\underbrace{2 \times 32 \times 2^{28}}_{\text{filling } V_1 \text{ and } V_2} + \underbrace{2 \times 28 \times 2^{28}}_{\text{sorting } V_1 \text{ and } V_2} + \underbrace{2^{-8} \times 2 \times 64 \times 2^{28}}_{\text{filling for the 32 pairs}} \Big)$$

which is equal to $2^{51.95}$ operations.

The memory complexity is dominated by the space required to store V_1 and V_2. Since each table contains 2^{28} rows of 64 bits, each core stores $2^{28} \times 64 \times 2 = 2^{35}$ bits. The 64 cores store a total of 2^{41} bits, which are 256 GB.

5.3 Details on the Practical Execution of the Attack

The program was written in C++ and it was executed on 64 cores clocked at 2,10 Ghz on an Intel Xeon CPU E5-2695. The result of Algorithm 1 was returned after less than 2 days (a total of 114 CPU days). As said previously 256 GB of memory were required.

Implementation choices. The time complexity of the attack being around 2^{52} simple operations we needed to optimize the implementation. We made two main choices:

[7] Sorting a table of n elements requires roughly $n \log n$ operation so we consider that the complexity of this step is equal to 28×2^{28}.

Algorithm 1: Improved recovery of tk_{11}

```
Input: 2^5 pairs of messages following the truncated differential path
Output: Last round tweakey tk11.
for tk11[7] ← 0 to f do
    for tk10[6] ← 0 to f do
        for tk11[3] ← 0 to f do
            for tk9[2] ← 0 to f do
                Initialise a table V2
                for all possible values of tk11[0], tk11[5], tk10[1], tk10[4],
                  tk11[6], tk9[7] and tk8[3] do
                    compute the difference in X7[12] for the first 2^4 pairs and
                      store it in V2
                end
                Initialise a table V1
                for all possible values of tk11[4], tk10[0], tk10[7], tk11[2],
                  tk11[1], tk9[5] and tk8[4] do
                    compute the difference in X7[0] for the first 2^4 pairs and
                      store it in V1
                end
                Sort V1 and V2
                Merge V1 and V2 on the 2^4 values of X7[12] and X7[0]
                if the intersection is not empty then
                    repeat the algorithm for 2^5 pairs (the same 2^4 plus 2^4
                      new), passing the matching value as parameter
                    only one guess of tk11 survives, return this one
                else
                    the guess for tk11[7], tk10[6], tk11[3] and tk9[2] is incorrect
                end
            end
        end
    end
end
```

1. It is possible to decrease the memory requirement to only $2^{7\times 4}\times 64 = 2^{34}$ bits by making the code parallel only once the 4 nibbles $tk_9[2]$, $tk_{10}[6]$, $tk_{11}[3]$ and $tk_{11}[7]$ are set. However, we found experimentally that this is much slower than making the whole code parallel. Indeed, filling V_i requires only $2 \times 32 \times 2^{28} = 2^{34}$ simple operations and using 64 cores for this is not worth.

2. It seems possible to save some time complexity by using an hash table instead of two sorted arrays. While this is true in theory, in practice we found the sorted arrays to be much faster. The main reason is the non-sequential accesses to the hash table, which are likely to lead to a cache miss at each access due to its size.

Once we have the full round tweakey tk_{11} we decrypt the last round and launch a similar attack on 11 rounds (with complexities that are negligible in comparison to the 12-round one) an so on until the master key is recovered.

5.4 On the Choice of the Distinguisher

As detailed in Section 3, another distinguisher is available over 7 rounds of SKINNY-64-128, corresponding to the equality of the difference in one nibble of the second row with the difference in the nibble in the last row and same column (see Property 2 in Figure 4). However, this option would have lead to an heavier key-recovery step as more nibbles of the key need to be guessed to obtain the required differences at the end of the distinguisher. More generally, our attack scenario can be played with different parameters as one can change the active nibble of the truncated characteristic and/or nibbles involved in the distinguisher. Hence we used the tool devised in the Crypto 16 article *Automatic Search of Meet-in-the-Middle and Impossible Differential Attacks* by Patrick Derbez and Pierre-Alain Fouque [4] to exhaust all cases and keep the one leading to the smallest complexity.

6 Attacking 10-round SKINNY-128-128

In Section 4, we found 9 structures of 4 plaintexts each such that the cell number 9 sums to zero over the 6-round SKINNY-128-128 encryptions of those plaintexts. These quadruples of plaintexts differ in two bytes and sum to zero. In other words, these quadruples form a second-order difference with two active bytes and the described distinguisher is a second-order truncated differential distinguisher[8] (see [8,7]). This distinguisher can be defined for arbitrary differences $\alpha, \beta \in \mathbb{F}_2^8$ as

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \end{bmatrix} \xrightarrow{6 \text{ rounds}} \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & B & ? & ? \\ ? & ? & ? & ? \end{bmatrix},$$

where $\otimes$ means that the second-order differential is formed by the two operands as the basis, B means that the cell sums to zero and ? means that the sum is unspecified. That is, the last two bytes must have differences $(0,0), (\alpha,0), (0,\beta), (\alpha,\beta)$ from one of the plaintexts in the quadruple, for any $\alpha, \beta \in \mathbb{F}_2^8$. A graphical representation of the truncated path of this distinguisher is given in Figure 9.

6.1 Key-Recovery using Truncated Differentials

The key recovery step is rather straightforward. The cell number 9 after 6 rounds of SKINNY-128-128, denoted $X_6[9]$, can be computed from the 10-round ciphertext using only 6 bytes of the key. Let I denote the inverse of the SKINNY-128

[8] Such distinguishers are also called *integral* distinguishers.

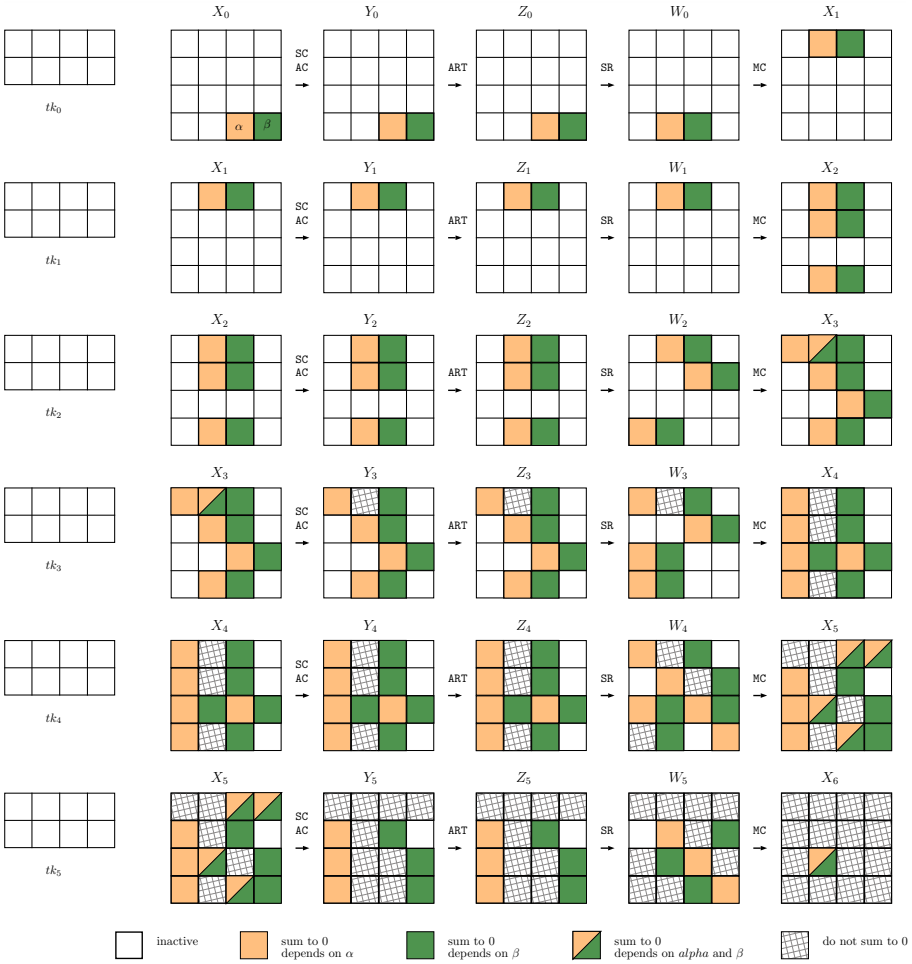

Fig. 9. Distinguisher on 6 rounds used in the 10-round attack on SKINNY-128-128.

S-Box, $C \in (\mathbb{F}_2^8)^{16}$ denote the ciphertext and $k \in (\mathbb{F}_2^8)^{16}$ denote the master key. Then

$$
\begin{aligned}
&X_6[9] = I(I(k_{12} \oplus a \oplus b \oplus c) \oplus I(d \oplus e)), \text{ where}\\
&a = I(3 \oplus k_7 \oplus I(k_9 \oplus C_6 \oplus C_{10} \oplus C_{14}) \oplus I(C_7 \oplus C_{15}) \oplus I(C_0 \oplus C_{12})),\\
&b = I(2 \oplus I(k_{15} \oplus C_7 \oplus C_{11} \oplus C_{15}) \oplus I(C_1 \oplus C_{13})),\\
&c = I(I(k_8 \oplus C_7) \oplus I(C_2 \oplus C_{14})),\\
&d = I(k_3 \oplus I(k_{15} \oplus C_7 \oplus C_{11} \oplus C_{15})),\\
&e = I(I(k_{12} \oplus C_5) \oplus I(C_0 \oplus C_{12})).
\end{aligned}
$$

It follows that $X_6[9]$ depends only on 6 key bytes:

$$k_3, k_7, k_8, k_9, k_{12}, k_{15}.$$

As can be seen in Figure 10, we again benefit from the tweakey schedule as the tweakey byte required in tk_7 is equal to one of the 4 bytes guessed in tk_9, namely the byte k_{12} of the master key.

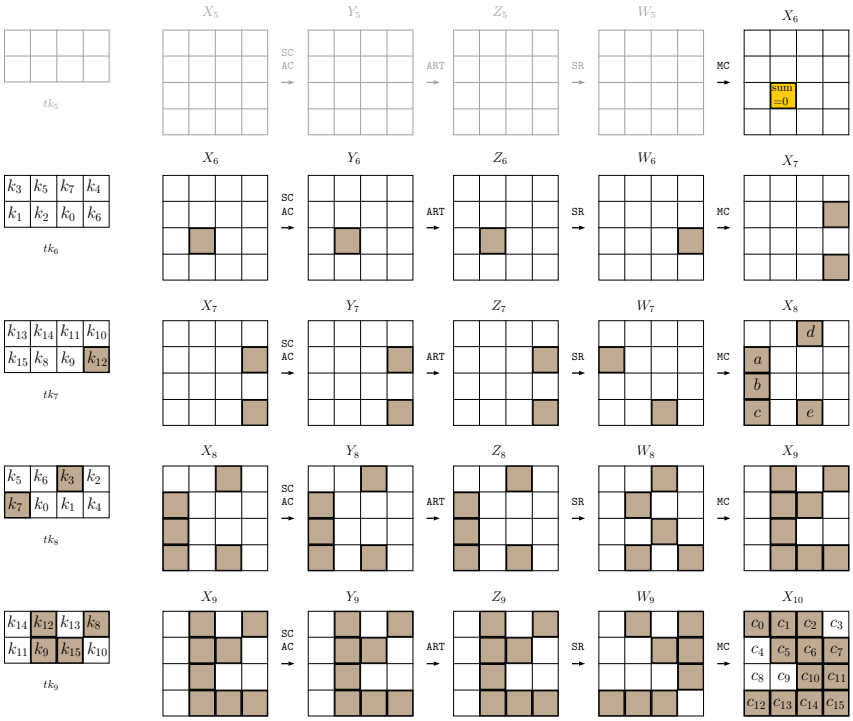

Fig. 10. Key recovery step used in the 10-round attack on SKINNY-128-128.

It is enough to consider 6 out of 9 quadruples following the trail in order to obtain a filter of probability 2^{-48}. Then, the 6 involved bytes of the key can be found by exhaustive search. The check consists in computation of $X_6[9]$ from ciphertexts using the equation above and verifying that the sum is equal to zero.

The rest of the key can be recovered in a similar way, but with complexity negligible compared to the main attack.

6.2 Details on the Practical Execution of the Attack

We implemented the attack in `C` language and executed it on 8 cores of an Intel Xeon CPU E5-2637 clocked at 3.50GHz. It exhausted the search space in 4 days (at total of 32 CPU days) and successfully determined the right key.

In the implementation, we performed the following optimisation. The computation of $X_6[9]$ can be structured as a sequence of four table lookups of size at most 2^{24} with negligible precomputation time and less than $6 \times 4 \times 2^{24}$ bytes of extra memory. For each ciphertext C, we precompute mappings $t_1, t_2, t_4 : \mathbb{F}_2^{24} \to \mathbb{F}_2^8, t_3 : \mathbb{F}_2^{16} \to \mathbb{F}_2^8$ such that $X_6[9]$ can be computed from the master key bytes $k_7, k_8, k_9, k_3, k_{12}, k_{15}$ as follows:

$$
\begin{aligned}
t_1 &= t_1(k_7, k_8, k_9) = a \oplus c, \\
t_2 &= t_2(k_3, k_{12}, k_{15}) = I(d \oplus e), \\
t_3 &= t_3(k_{12}, k_{15}) = k_{12} \oplus b, \\
t_4 &= t_4(t_1, t_2, t_3) = I(I(t_1 \oplus t_3) \oplus t_2), \\
X_6[9] &= t_4.
\end{aligned}
$$

The final complexity of the attack is dominated by verifying the zero-sum property for one of the quadruples, which requires 4 computations of $X_6[9]$, i.e. 16 24-bit table lookups. The total complexity is thus 2^{52} table lookups. The memory complexity is equal to $(2 \times 6 + 1) \times 2^{24} + 6 \times 2^{16}$ bytes $\approx$ 208 MB (note that t_4 does not depend on the ciphertext) and can be further reduced if necessary.

6.3 On the Choice of the Distinguisher

We remark that choosing another distinguishers from Table 3 would lead to similar or higher complexities.

The properties $\bigoplus c_i$ for $8 \leq i \leq 11$ are equivalent in the sense that corresponding cells c_i can be computed from the ciphertext using only 6 key bytes. However, for the case $i = 9$ there are more structures in the provided data set.

All properties $\Delta c_i = \Delta c_j$ from Table 3 lead to similar or higher time complexity of the key recovery. Indeed, for each of these properties at least one of the two cells c_i, c_j requires at least 6 key bytes to be guessed to compute the corresponding difference. For example, consider the property $\Delta c_0 = \Delta c_{12}$ after 6 rounds (similar to the 7-round distinguisher used to attack `SKINNY`-64-128). The cell c_0 can be computed from the ciphertext using 5 key bytes $k_0, k_3, k_{10}, k_{14}, k_{15}$; the cell c_{12} can be computed from the ciphertext using 6 key bytes $k_3, k_4, k_{10}, k_{11}, k_{12}, k_{15}$. The attacker first guesses 3 common key bytes k_3, k_{10}, k_{15}. Then she enumerates all values of k_0, k_{14}, computes the differences Δc_0 and puts them in a table. Finally, she enumerates all values of k_4, k_{11}, k_{12}, computes the differences Δc_{12} and checks if they are in the table. This approach requires $O(2^{48})$ time and $O(2^{16})$ memory.

7 Conclusion

In this cryptanalysis report we detailed practical key recovery attacks of 12-round SKINNY-64-128 and 10-round SKINNY-128-128 from given sets of 2^{20} messages. Our attacks consist in leveraging a distinguisher based on a probability-1 truncated first-order and second-order differential paths. The attacks are possible because the provided sets of messages give much more exploitable pairs than what one could have expected from a random set.

References

1. Ankele, R., Banik, S., Chakraborti, A., List, E., Mendel, F., Sim, S.M., Wang, G.: Related-key impossible-differential attack on reduced-round skinny. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17. LNCS, vol. 10355, pp. 208–228. Springer, Heidelberg (Jul 2017)
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: SIMON and SPECK: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585 (2015), http://eprint.iacr.org/2015/585
3. 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 and Katz [10], pp. 123–153
4. Derbez, P., Fouque, P.A.: Automatic search of meet-in-the-middle and impossible differential attacks. In: Robshaw and Katz [10], pp. 157–184
5. Jean, J.: TikZ for Cryptographers. https://www.iacr.org/authors/tikz/ (2016)
6. 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. LNCS, vol. 8874, pp. 274–288. Springer, Heidelberg (Dec 2014)
7. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE'94. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (Dec 1995)
8. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography: Two Sides of One Tapestry. pp. 227–233. Springer US, Boston, MA (1994), https://doi.org/10.1007/978-1-4615-2694-0_23
9. Liu, G., Ghosh, M., Song, L.: Security analysis of SKINNY under related-tweakey settings (long paper). IACR Trans. Symm. Cryptol. 2017(3), 37–72 (2017)
10. Robshaw, M., Katz, J. (eds.): CRYPTO 2016, Part II, LNCS, vol. 9815. Springer, Heidelberg (Aug 2016)

Appendix

Table 4. 4-bit Sbox used in SKINNY-64.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}_4(x)$	c	6	9	0	1	a	2	b	3	8	5	d	4	e	7	f

Table 5. 8-bit Sbox used in SKINNY-128.

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

Algorithm 2: Recovery of the tweakey nibbles involved in the computation of U_1 and U_2

Input: Pairs of messages following the truncated differential path
Output: Last round tweakey tk_{11}, nibbles 0, 1, 4, 6 and 7 of tk_{10}, nibbles 2, 5 and 7 of tk_9, nibbles 3 and 4 of tk_8 (and 0 of tk_7).

```
Initialise two tables T_1 and T_2 and compute X_11[8-15] and Z_11[0-7] from the ciphertexts
for tk_11[7] ← 0 to f do
    compute X_11[7]
    for tk_10[6] ← 0 to f do
        for tk_11[3] ← 0 to f do
            compute X_10[6], X_11[3] and X_10[12]
            for tk_9[2] ← 0 to f do
                compute X_9[2]
                for tk_11[0] ← 0 to f do
                    compute X_11[0] and X_10[13]
                    for tk_11[5] ← 0 to f do
                        compute X_11[5]
                        for tk_10[1] ← 0 to f do
                            compute X_10[1], X_9[14] and X_8[15]
                            for tk_10[4] ← 0 to f do
                                compute X_10[4]
                                for tk_11[6] ← 0 to f do
                                    compute X_11[6] and X_10[8]
                                    for tk_9[7] ← 0 to f do
                                        compute X_9[7]
                                        for tk_8[3] ← 0 to f do
                                            compute the difference in X_7[12] for all
                                              the pairs and store it in T_2 next to the
                                              key guesses
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
                for tk_11[4] ← 0 to f do
                    compute tk_7[0] by using Equation (1)
                    compute X_11[4] and X_10[10]
                    for tk_10[0] ← 0 to f do
                        compute X_10[0] and X_9[13]
                        for tk_10[7] ← 0 to f do
                            compute X_10[7]
                            for tk_11[2] ← 0 to f do
                                compute X_11[2], X_10[15] and X_9[9]
                                for tk_11[1] ← 0 to f do
                                    compute X_11[1] and X_10[14]
                                    for tk_9[5] ← 0 to f do
                                        for tk_8[4] ← 0 to f do
                                            compute the difference in X_7[0] for all the
                                              pairs and store it in T_1 next to the key
                                              guesses
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
                Merge T_1 and T_2 and return the corresponding keys
            end
        end
    end
end
```