



# Meet-in-the-Middle Attacks and Structural Analysis of Round-Reduced PRINCE\*

Patrick Derbez

Univ Rennes, CNRS, IRISA, Rennes, France  
patrick.derbez@irisa.fr

Léo Perrin

Inria, Paris, France  
leo.perrin@inria.fr

Communicated by Vincent Rijmen.

Received 31 March 2016 / Revised 12 February 2020  
Online publication 4 March 2020

**Abstract.** NXP Semiconductors and its academic partners challenged the cryptographic community with finding practical attacks on the block cipher they designed, PRINCE. Instead of trying to attack as many rounds as possible using attacks which are usually impractical despite being faster than brute force, the challenge invites cryptographers to find practical attacks and encourages them to actually implement them. In this paper, we present new attacks on round-reduced PRINCE including the ones which won the challenge in the 4-, 6- and 8-round categories—the highest for which winners were identified. Our first attacks rely on a meet-in-the-middle approach and break up to ten rounds of the cipher. We also describe heuristic methods we used to find practical SAT-based and differential attacks. Finally, we also present an analysis of the cycle structure of the internal rounds of PRINCE leading both to a low complexity distinguisher for 4-round PRINCE-core and an alternative representation of the cipher valid in particular contexts and which highlights, in these cases, a poor diffusion.

**Keywords.** PRINCE, Practical attacks, Meet-in-the-middle, SAT-solver, Statistical analysis.

## 1. Introduction

When tasked with assessing the security of a block cipher, cryptanalysts have now a broad range of tools at their disposal: differential attack [1], linear attack [2], meet-in-

---

\*Patrick Derbez and Léo Perrin were supported by the CORE ACRYPT project from the *Fond National de Recherche* (Luxembourg).

This article is the full version of the article with the same title published at Fast Software Encryption 2015.

the-middle attack [3], etc. The main purpose of a security analysis is usually to identify flaws in the design of a primitive and then to illustrate their gravity through the description of an attack covering as many rounds as possible. However, applicability of said attacks in a realistic situation is usually not the first objective of the cryptanalyst. A simple reason for this is that as our understanding of the design of block ciphers improved, the ease of identifying practical attacks decreased. Furthermore and in accordance with the famous maxim “attacks only get better,” an impractical attack submitted at a given time may later be improved.

While impractical attacks provide the academic community with valuable insights into the security provided by different block ciphers, their components, their design strategies, etc., cryptanalysis in the industry is more focused on practical attacks. In order to promote this view, the Technical University of Denmark (DTU), NXP Semiconductors and the Ruhr University of Bochum challenged the cryptographic community [4] with finding low data complexity attacks on the block cipher PRINCE [5]. More precisely, they accept attacks requiring only at most  $2^{20}$  chosen plaintexts or  $2^{30}$  known plaintexts. Furthermore, extra rewards (from 1000 to 10000€) are given for attacks on at least eight rounds which require at most  $2^{45}$  bytes of memory (about 32 Terabytes) and at most  $2^{64}$  encryptions of the round-reduced variant attacked.

Studying PRINCE in this setting may provide valuable data on multiple accounts. First of all, PRINCE is a lightweight block cipher, meaning that it is intended to be run on processors with little computing power to devote to security-related algorithm or on hardware where every logical gate counts. Research on this topic is intense nowadays as the need for such primitives becomes increasingly pressing, see [6] for an extensive review of the algorithms that have been proposed. Second, PRINCE implements a simplified version of the so-called FX construction: Encryption under key  $(k_0||k_1)$  consists in XOR-ing  $k_0$  to the plaintext, applying a block cipher called PRINCE-core keyed with  $k_1$ , and then, output the result XOR-ed with  $L(k_0)$  where  $L$  is a simple linear bijection. This strategy allows for a greater key size without the cost of a sophisticated key schedule. However, it is impossible to make a security claim as strong as for a more classical construction. Finally, PRINCE-core has a unique property called  $\alpha$ -reflection. If we denote by  $E_{k_1}^c$  the encryption under PRINCE-core with subkey  $k_1$ , then the corresponding decryption operation is  $E_{k_1 \oplus \alpha}^c$  for a constant  $\alpha$ . In other words, decryption is merely encryption under a related key. The consequences of this property have already been studied, and, in particular, some values of  $\alpha$  different from the one used have been shown to lead to weaker algorithms [7].

PRINCE has already been the subject of several cryptanalyses, notably [8] where the security of the algorithm against multiple attacks was assessed, [7] which investigated the influence of the value of  $\alpha$ , [9] which described meet-in-the-middle attacks on the block cipher and, finally, [10] which proposed the best attack to date in terms of number of rounds attacked. A list of the cryptanalyses of round-reduced PRINCE is provided in Table 1 (attacks working only on PRINCE-core or for modified versions of PRINCE (different  $\alpha$  and/or S-Box) are not shown in this table).

As stated before, most of the attacks usually considered often have impractical complexities. For instance, differential attacks and linear attacks require large amounts of chosen (respectively, known) plaintexts, both of which may be impossible to gather to begin with if the algorithm is implemented on a small device with little computer and,

**Table 1.** The best attacks on round-reduced PRINCE in the single-key model.

Description			Complexity		
References	Type	Rounds	Data (CP)	Time	Memory
[11]	Integral	4	$5 \times 2^4$	$5 \times 2^8$	$2^4$
		5	$6 \times 2^4$	$2^{29}$	$2^4$
		6	$6 \times 2^{16}$	$2^{41}$	$2^{16}$
Sect. 5	Bit-pattern integral	4	$6 \times 2^3$	$2^{28}$	$2^3$
	Higher-order differential	7	$6 \times 2^{57}$	$2^{57}$	$2^{57}$
"Appendix D"	MitM	4	$2^{10}$	5s	$\ll 2^{27}$
		4	33 KP	$2^{43.4}$	$2^{26.7}$
		6	64 KP	$2^{101.1}$	$2^{34}$
Sect. 3		8	$2^{16}$	$2^{66.25}$	$2^{49.9}$
		6	$2^{16}$	$2^{33.7}$	$2^{31.9}$
		8	$2^{16}$	$2^{50.7}$ (online)	$2^{84.9}$
		8	$2^{16}$	$2^{65.7}$ (online)	$2^{68.9}$
Sect. 4	Differential	10	$2^{57}$	$2^{68}$ (online)	$2^{41}$
		6	$2^{14.9}$	$2^{25.1}$	$2^{14.9}$
[9]	MitM	8	$2^{53}$	$2^{60}$	$2^{30}$
		9	$2^{57}$	$2^{64}$	$2^{57.3}$
[10]	Multiple diff.	9	$2^{46.89}$	$2^{51.21}$	$2^{52.21}$
		10	$2^{57.94}$	$2^{60.62}$	$2^{61.52}$

Time complexity is measured in encryption units

Memory complexity is measured in 64-bit blocks

hence, a small throughput. Therefore, we focused our efforts on meet-in-the-middle (MitM) attacks, algebraic/logic attack where the fact that a ciphertext is the encryption of a plaintext is encoded as an equation which is fed to a solver and, surprisingly, differential attack for which we found a heuristic method decreasing significantly the data complexity.

*Our contribution* We describe different low data complexity attacks on round-reduced PRINCE which were submitted to the PRINCE challenge and turned out [12] to be the best ones on PRINCE reduced to four, six and eight rounds. In Sect. 3, we describe our attacks obtained using the meet-in-the-middle technique and we also show a new attack on 10 rounds with practical memory and a time complexity around  $2^{68}$  encryptions. Then, we describe in Sect. 4 a low data differential attack against 6-round PRINCE, which is the fastest known. In fact, the power of the filter used to discard wrong pairs in a differential attack can be raised to the power 4 when attacking 6-round PRINCE by considering groups of pairs. In Sect. 5, we show how the equation given to a SAT-solver can be modified so as to make an attack on four rounds practical, and how to simplify our 6-round differential attack by using a SAT-solver to take care of the tedious details of the key recovery. We finally present in Sect. 6 some observations about the cycle structure of the internal rounds of PRINCE and how it implies the existence of alternative representations of the cipher highlighting a poor diffusion in some subsets of the input space. While we do not use these to attack PRINCE directly, we show that the size of these subsets remains reasonable and actually find such sets for 4-round PRINCE-core.

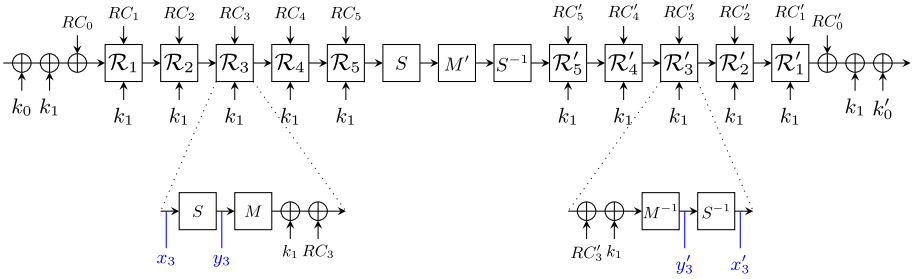


Fig. 1. The PRINCE cipher.

Extra materials including codes of practical attacks can be found at: <https://seafife.cifex-dedibox.ovh/d/ccd77d431e5c4ea28de1/>.

## 2. Specification of PRINCE

### 2.1. Description of PRINCE

PRINCE is a 64-bit block cipher with a 128-bit key. It is based on a variant of the FX construction which was proposed by Kilian and Rogaway as a generalization of the DESX scheme. The master key  $k$  is split into two 64-bit parts  $k = k_0 \parallel k_1$ , and  $k_0$  is used to generate a third subkey  $k'_0 = (k_0 \ggg 1) \oplus (k_0 \ggg 63)$ . Both  $k_0$  and  $k'_0$  are used as pre- and post-whitening keys, respectively. The full version of the cipher has 12 rounds and is depicted in Fig. 1.

The encryption is quite similar to the AES and consists of a nibble-based substitution layer  $S$  and a linear layer  $M$ . The operation  $M$  can be divided into a ShiftRows operation and a matrix multiplication  $M'$  operating independently on each column but not nibble oriented. Furthermore, the matrix  $M'$  is an involution, and combined to the fact that the round constants satisfy the relation  $RC_i \oplus RC'_i = \alpha$  where  $\alpha = \text{C0AC29B7C97C50DD}$ , the decryption process  $D_{k_0, k_1, k'_0}$  is equal to the encryption process  $E_{k'_0, k_1 \oplus \alpha, k_0}$ . For further details about PRINCE, we refer the reader to [5].

*Notations* In the sequel, we denote both the plaintext and the ciphertext by  $p$  and  $c$ , respectively. For the first  $R$  rounds of  $2R$ -round PRINCE, we denote the internal state just before (resp. after) the  $r$ -th SubNibble layer by  $x_r$  (resp.  $y_r$ ), while for the last  $R$  rounds, those internal states are denoted by  $y'_r$  and  $x'_r$ , respectively, as shown in Fig. 1. Given a collection of messages  $\{p^0, \dots, p^m, \dots\}$ ,  $x_r^m[i]$  represents nibble  $i$  of state  $x_r$  of message  $p^m$ . As PRINCE is not fully nibble oriented, we use the notation  $x_r[i]_b$  to refer to the bit  $i$  of the state  $x_r$  and the following relation holds for all  $i \in \{0, \dots, 15\}$  (Fig. 2):

$$x_r[i] = x_r[4i + 3]_b \parallel x_r[4i + 2]_b \parallel x_r[4i + 1]_b \parallel x_r[4i]_b.$$

Finally, we use the following notations for some functions:

$R$ : the composition of  $S$  and  $M$  so that  $R(x) = M(S(x)) = SR(M'(S(x)))$ .

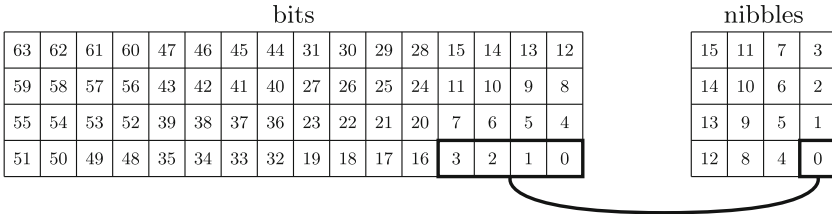


Fig. 2. Ordering of bits/nibbles in PRINCE.

- $E_{k_0||k_1}^r$  : PRINCE reduced to  $r$  rounds.
- $E_{k_1}^c$  : full PRINCE-core.
- $E_{k_1}^{c,r}$  : PRINCE-core reduced to  $r$  rounds.

### 3. Meet-in-the-Middle Attacks

In this section, we present both the 6-round attack and the 8-round attack which won the PRINCE challenge in the chosen-plaintext category together with a new attack on ten rounds. The aim of the challenge was to find the best attacks using at most  $2^{20}$  chosen plaintexts, and thus, we decided to follow the strategy used by Demirci and Selçuk on AES in [3], later improved by Dunkelman et al. in [13], Derbez et al. in [14, 15] and by Li et al. in [9]. While our 10-round attack does not fit the restriction on the data complexity, it shows that this kind of attacks is one of the most powerful on SP network.

First, we give the definition of an ordered  $\delta$ -set which is a particular structure of messages used in our attacks.

**Definition 1.** Let a  $\delta$ -set be a set of 16 PRINCE states that are all different in one state nibble (the active nibble) and all equal in the other state nibble (the inactive nibbles). An ordered  $\delta$ -set is a  $\delta$ -set  $\{x^0, \dots, x^{15}\}$  such that the difference in the active nibble between  $x^0$  and  $x^i$  is equal to  $i$ , for  $0 \leq i \leq 15$ .

In the sequel, we consider  $\delta$ -sets such that nibble 7 is the active one. For such a particular set, we made the following observations which are the core of our new attacks.

**Observation 1.** Consider the encryption of a collection  $\{p^0, p^1, \dots, p^{15}\}$  of 16 messages through 6-round PRINCE. If the set  $\{y_2^0, y_2^1, \dots, y_2^{15}\}$  is an ordered  $\delta$ -set, then the ordered sequence

$$\left[ y_2^1[7] \oplus y_2^0[7], y_2^2[7] \oplus y_2^0[7], \dots, y_2^{15}[7] \oplus y_2^0[7] \right]$$

is fully determined by the following eight nibble parameters:

$$-x_3^0[0, 7, 10, 13] \qquad -x_3^0[0, 7, 10, 13]$$

Consequently, there are at most  $2^{8 \times 4} = 2^{32}$  possible sequences when we consider all the possible choices of keys and ordered  $\delta$ -sets (out of the  $2^{4 \times 15} = 2^{60}$  of the theoretically possible 15-nibble sequences).

*Proof.* The proof is straightforward. The goal is to propagate the differences from the state  $y_2$  (which are known) to the state nibble  $y_2'[7]$ . At each intermediate round, each input of an S-Box is either a parameter, not required or constant (so output differences are equal to zero).  $\square$

**Observation 2.** Consider the encryption of a collection  $\{p^0, p^1, \dots, p^{15}\}$  of 16 messages through 8-round PRINCE. If the set  $\{x_2^0, x_2^1, \dots, x_2^{15}\}$  is an ordered  $\delta$ -set, then the ordered sequence

$$\left[ x_2'^1[7] \oplus x_2'^0[7], \dots, x_2'^{15}[7] \oplus x_2'^0[7], y_2'^1[6] \oplus y_2'^0[6], \dots, y_2'^{15}[6] \oplus y_2'^0[6] \right]$$

is fully determined by the following 42 nibble parameters:

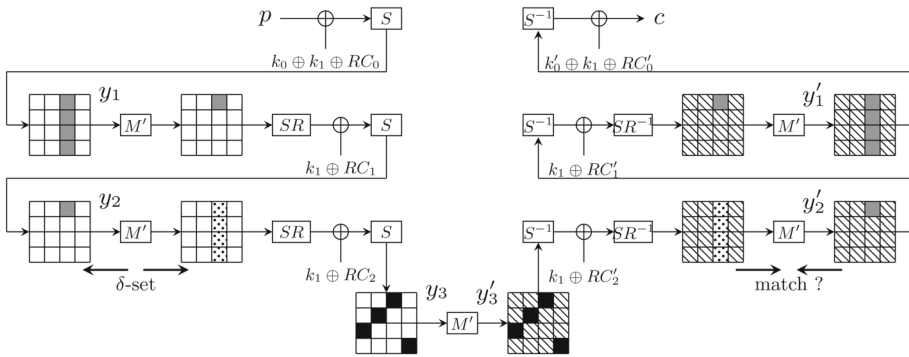
$$\begin{array}{ll} -x_2^0[7] & -x_4^0[0..15] \\ -x_3^0[0, 7, 10, 13] & -x_3^0[0, 7, 10, 13] \\ -x_4^0[0..15] & -x_2^0[7] \end{array}$$

Furthermore, those 42 state nibbles can be directly computed from the full state  $x_4$  and four nibbles of  $M^{-1}(k_1)$ . Consequently, there are at most  $2^{4 \times (16+4)} = 2^{80}$  possible sequences when we consider all the possible choices of keys and ordered  $\delta$ -sets (out of the  $2^{4 \times 30} = 2^{120}$  of the theoretically possible 30-nibble sequences).

*Proof.* The proof is similar to the one of Observation 1 except the parameters are related. Indeed, from the full state  $x_4$  one can directly compute  $x_4'$  as no keys are involved ( $x_4' = S^{-1}(M'(S(x_4)))$ ). Then, we note that the four nibbles  $M^{-1}(k_1)[4..7]$  are enough to compute  $x_3^0[0, 7, 10, 13]$  from  $x_4$  and  $x_3^0[0, 7, 10, 13]$  from  $x_4'$ . Finally, the knowledge of  $M^{-1}(k_1)[7]$  allows to compute  $x_2^0[7]$  and  $x_2^0[7]$  from  $x_3^0[0, 7, 10, 13]$  and  $x_3^0[0, 7, 10, 13]$ , respectively.  $\square$

### 3.1. 6-Round Attack

The 6-round attack is depicted in Fig. 3, and its scenario is straightforward. First, the  $2^{32}$  possible sequences given in Observation 1 are computed and stored in a hash table during a preprocessing phase. Then during the online phase, we begin by asking for the encryption of a structure of  $2^{16}$  chosen plaintexts such that nibbles from 4 to 7 take all the possible values, while the other ones are constant, and pick one of them denoted  $p^0$ . Now, the goal of the adversary is to identify an ordered  $\delta$ -set containing  $y_2^0$ . To do so, he has to guess the five nibbles  $x_1^0[4..7]$  and  $x_2^0[7]$  and propagate the differences from the state  $y_2$  to the plaintext. Then, he gets the corresponding ciphertexts, guesses the five nibbles  $x_1^0[4..7]$  and  $x_2^0[7]$  and propagates the differences from the ciphertexts to  $y_2'[7]$ . Finally, he discards all the guesses which do not lead to a match in the previously built hash table. The probability for a wrong guess to pass the test is  $2^{32} \times 2^{-60} = 2^{-28}$ ,



**Fig. 3.** 6r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

so we expect  $2^5$  candidates to remain at the end of the attack. The wrong ones can be discarded by replaying the attack with an other choice for  $p^0$  without increasing the overall complexity of the attack.

The data complexity of this attack is  $2^{16}$  chosen plaintexts, and the memory requirement is around  $2^{32} \times 4 \times 15 \times 2^{-3} \approx 2^{34.9}$  bytes. During the online phase, ten state nibbles are guessed; however, they can assume only  $2^{33}$  values once the plaintext/ciphertext pair is given. Indeed, the knowledge of the 33 bits:

$$\{(k_0 \oplus k_1)[16..27]_b, (k'_0 \oplus k_1)[16..27]_b, k_1[28..31]_b, k_0[28..32]_b\},$$

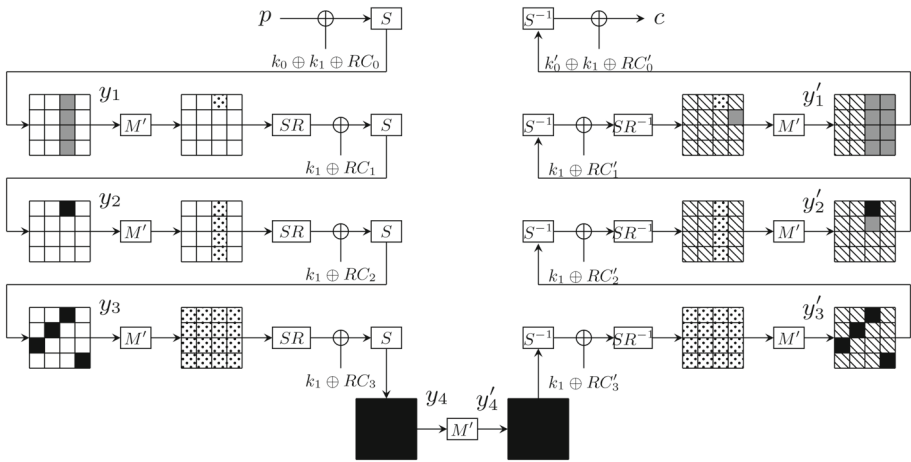
is enough to compute all of them from  $p$  and  $c$ . Thus, the time complexity of the online phase is approximately  $16 \times 2^{33} \times 40 / (6 \times 64) \approx 2^{33.7}$  encryptions.

*Key recovery* At the end of the attack,  $128 - 33 = 95$  key bits are still missing. To find them, the best way is to apply several meet-in-the-middle attacks successively. For instance, one could begin by running the attack depicted in Fig. 12 in “Appendix A” which has an overall complexity below  $2^{28}$  as most key bits required in the online phase are already known.

### 3.2. 8-Round Attack

The 8-round attack is similar to the one on six rounds and is depicted in Fig. 18. It relies on Observation 2, so the memory complexity is around  $2^{80} \times 15 \times 8 \times 2^{-3} \approx 2^{83.9}$  bytes. In the online phase, the data complexity remains unchanged to  $2^{16}$  chosen plaintexts, but the number of state variables to guess is increased. The identification step requires to guess the four nibbles  $x_1^0[4..7]$ , and then, the nine nibbles  $x_1^0[0..7]$  and  $x_2^0[6]$  are guessed to build the sequence from the ciphertexts. Those 13 nibbles can assume only  $2^{49}$  values once the plaintext/ciphertext pair  $(p^0, c^0)$  given as they all can be derived from

$$\{(k_0 \oplus k_1)[16..24, 28..31]_b, (k'_0 \oplus k_1)[0..23, 27..31]_b, k_0[25..27]_b, k_1[24..27]_b\}.$$



**Fig. 4.** 8r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

Thus, the time complexity of the online phase is approximately  $16 \times 2^{49} \times 52 / (8 \times 64) \approx 2^{49.7}$  encryptions and we expect  $2^{49} \times 2^{80} \times 2^{-120} = 2^9$  candidates to remain at the end of the attack (Fig. 4).

*Key recovery* As for the previous attack, the most efficient way to recover the missing key bits is to perform other attacks. For instance, one could run the attack depicted in Fig. 13 (“Appendix B”) which has the same complexity than the one above since there are approximately  $2^9$  candidates for the four active nibbles of  $x_1$ . Then, the search space would be small enough to perform an exhaustive search without increasing the overall complexity.

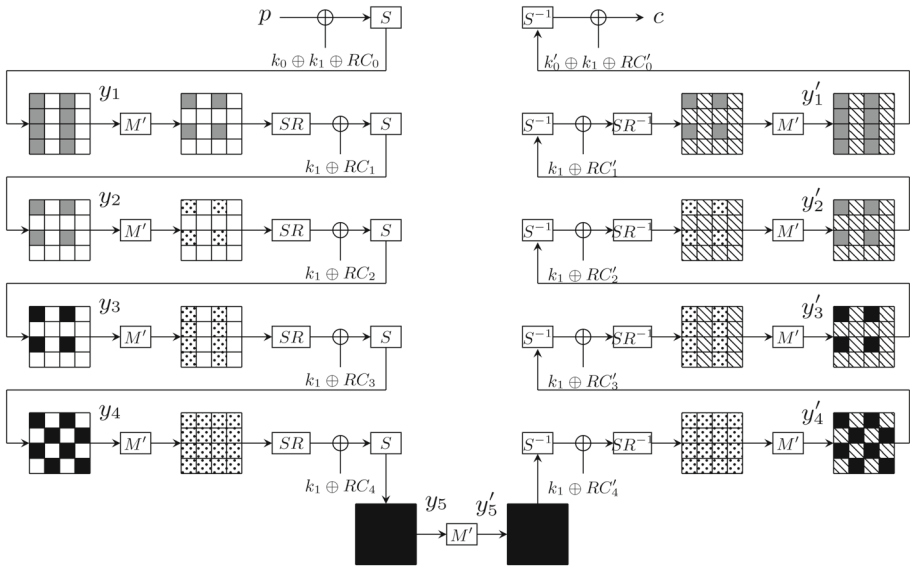
*Trade-off* It is possible to trade some memory against time without increasing the data complexity by noticing that for a considered structure of  $2^{16}$  plaintexts, the four active nibbles of  $x_3$  take all the possible values. Thus, we can fix them to 0 during the offline phase and save a factor  $2^{16}$  in memory. In the other hand, we now need to run the attack for all the possible choices for  $p^0$  increasing the time complexity by the same factor of  $2^{16}$ .

A more sophisticated attack against 8-round PRINCE, requiring much less memory, is described in “Appendix E.”

### 3.3. 10-Round Attack

We now investigate PRINCE reduced to ten rounds. While we were unable to find an attack requiring less than  $2^{20}$  chosen plaintexts for the PRINCE challenge, we found one competitive with the current best known attack. To describe it, we first extend the definition of a  $\delta$ -set as it was done in [14], then we show a meet-in-the-middle attack as the two ones above and finally we apply the differential enumeration technique ([13]).  $\delta$ -set. In [14], Derbez *et al.* showed that the notion of  $\delta$ -set can be extended to set of states such that some *linear combinations* of state bits are constant. In the sequel, we denote by





**Fig. 5.** 10r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

$\delta$ -set a set of 16 messages such that  $y_2[0..4, 6, 8..12, 14]$  and  $M'(y_2)[0..4, 6, 8..12, 14]$  are constant, exploiting the fact that the matrix operating on the columns is not MDS.

*10-round attack* The basis of our attack on 10 rounds is depicted in Fig. 5. The meet-in-the-middle is performed on the four-bit equations described above. The state bytes required as the parameters of the hash table can be computed from the whole state  $x_5$  and eight nibbles of the equivalent subkey  $M^{-1}(k_1)$ , and thus, approximately  $2^{96}$  60-bit sequences are stored. In the online phase, the 24 state nibbles needed can be computed from the following 66 key bits:

$$\begin{aligned} &\{k_0[0, 20..24, 28..32, 52..56, 60..63]_b, k_1[20..23, 28..31, 52..55, 60..63]_b, \\ &(k_0 \oplus k_1)[16..19, 24..27, 48..51, 56..59]_b, \\ &(k'_0 \oplus k_1)[16..19, 24..26, 48..51, 56..58]_b\}. \end{aligned}$$

Note that this attack does not actually work because the number of sequences stored is higher than the number of possible 60-bit sequences, and thus, no key candidates are filtered. The aim of the next section is to show how to reduce the memory requirement. *Differential Enumeration Technique* Li et al. applied this technique against PRINCE in [9] and successfully mounted new attacks on eight and nine rounds. The idea of this technique originally introduced by Dunkelman et al. in [13] is to store in the hash table only the sequences built from a  $\delta$ -set containing a message  $p^0$  that belongs to a pair  $(p^0, p^1)$  following a well-chosen differential characteristic. In our case, the truncated differential characteristic is depicted in Fig. 5 assuming a zero difference in hatched nibbles. Thus, we expect to store only  $2^{96+4-60} = 2^{40}$  sequences in the offline phase.

However, generating them is not as trivial as for the basic attack. We propose the following procedure which has a time complexity around  $2^{72}$  operations:

1. Consider a pair  $(p^0, p^1)$  following the differential characteristic.
2.  $S^{-1} \circ M' \circ S$  can be seen as four invertible super S-Boxes  $S_0, \dots, S_3$  operating on 16-bit words. Build four hash tables such that one can retrieve  $(x, y)$  from  $(x \oplus y, S_i(x) \oplus S_i(y))$ .
3. Guess the difference in the active nibbles of both  $y_4$  and  $y'_4$  and retrieve the actual value of  $x_5$  and  $x'_5$  for both messages of the pair.
4. Guess the difference in the two active nibbles of the first column of  $y_3$  and get back the actual values of  $y_4[2, 5, 8, 15]$ .
5. Combined with the knowledge of  $x_5$ , this leads to the knowledge of the four key nibbles  $M^{-1}(k_1)[2, 5, 8, 15]$ . Use them to partially encrypt  $x'_5$  and check whether the difference in the first column of  $y'_3$  is correct.
6. Use  $M^{-1}(k_1)[15]$  to partially decrypt  $y_4$  and get the difference in  $x_2[15]$  and check its correctness. Do the same for the difference in  $x'_2[15]$ .
7. Guess the difference in the two active nibbles of the third column of  $y_3$  and get back  $M^{-1}(k_1)[0, 7, 11, 13]$ .
8. Compute the value of the missing parameters and check whether the pair follows the characteristic or not. If it does, then build the 60-bit sequence from  $p^0$  and store it in the hash table.

The complexity of this procedure is dominated by the complexity of steps 4–5 which is  $2^{72}$  simple operations that we estimate to be equivalent to  $2^{69}$  encryptions. Now that the table is built the online phase is quite similar to the one of the offline phase:

1. Ask for a structure of  $2^{32}$  chosen plaintexts and store the ciphertexts in a hash table to identify the pairs that may follow the differential characteristic.
2. For each pair  $(p^0, p^1)$ :
  - (a) Guess the difference in the first column of  $y_1$  and of  $y_2$  and deduce the corresponding value of  $(k_0 \oplus k_1)[12..15]$  and  $k_1[15]$ . Store them in a hash table  $T_0$  indexed by  $k_1[15]$ ,  $k_0[61..63]_b$ .
  - (b) Similarly, compute  $(k'_0 \oplus k_1)[12..15]$  and  $k_1[15]$  from the ciphertexts and use  $T_0$  and the linear relations between  $k_0$  and  $k'_0$  to get back the  $2^{2 \times 4 + 2} \cdot 2^{-7} = 2^3$  corresponding values of the key nibbles above. Store those  $2^{13}$  key candidates in a hash table  $T_1$  indexed by  $(k_0 \oplus k_1)[12..15]$ ,  $(k'_0 \oplus k_1)[12..15]$  and  $k_0[55]_b \oplus k_0[60]_b (= (k_0 \oplus k_1)[55]_b \oplus \dots \oplus (k_0 \oplus k_1)[60]_b \oplus (k'_0 \oplus k_1)[55]_b \oplus \dots \oplus (k'_0 \oplus k_1)[59]_b \oplus k_1[60]_b)$ .
  - (c) Repeat the two steps above but now by guessing the third column of  $y_2$  and use  $T_1$  to obtain the  $2^{2 \times 13 - 8 - 8 - 1} = 2^9$  and store them in a hash table  $T_2$  indexed by the difference in  $y_2$ . (While the match is on 33 bits,  $(k_0 \oplus k_1)[12..15]$  and  $(k'_0 \oplus k_1)[12..15]$  only depend on four 4-bit parameters.)
  - (d) Repeat the three steps above but now by guessing the third column of  $y_1$  and use  $T_3$  to finally retrieve all the  $2^{9+9-8} = 2^{10}$  key candidates.
  - (e) For each key candidate, identify a  $\delta$ -set from  $p^0$ , build the 60-bit sequence and check whether it belongs to the table constructed in the offline phase. If it does, then try the key candidate.

3. Repeat the procedure until the right key is found.

As each structure contains  $2^{63}$  pairs and each of these pairs follows the differential with probability  $2^{-28-60} = 2^{-88}$ , we need  $2^{25}$  structures on average. Then, for each structure we have to study only  $2^{63-32} = 2^{31}$  pairs and for each of them we have to perform  $4 \times 2^{13} + 2^{10} \times 2^4$  simple operations estimated to approximately  $2^{12}$  encryptions. Thus, this procedure has the time complexity of  $2^{25+31+12} = 2^{68}$  encryptions and requires  $2^{25+32} = 2^{57}$  chosen plaintexts. At the end of the attack,  $2^{66} \times 2^{40} \times 2^{-60} = 2^{46}$  key candidates remain. As 62 key bits are also missing, performing an exhaustive search is not a valid option. Instead, the best way to recover the key is to apply several meet-in-the-middle attacks. For instance, we can assume that when a match happens, we get back the corresponding values of the red nibbles in Fig. 5 and then deduce step by step each key bits of  $M^{-1}(k_1)$  by completing the first and the third columns of  $y'_3$  without increasing the overall complexity of the attack.

### 4. Differential Attack

In this section, we describe a new differential attack against 6-round PRINCE. There have already been some differential cryptanalyses of PRINCE, see, for example, [10], which is the best attack to date, and also [16]. Our attack uses a new method to increase the power of the filter by considering groups of pairs. This allows a rather low data complexity considering that 6 rounds are attacked and that differential attacks usually demand large amounts of chosen plaintexts.

#### 4.1. Amplified Differential Trails

Our attack relies on some differences propagating identically in different pairs. To better describe this, we introduce the following definitions:

- Encryption We call *encryption* a couple plaintext/ciphertext encrypted under a fixed key.
- Pair A *pair* is a set of two encryptions where the plaintexts are separated by a known difference.
- Family A *family* is a group of pairs with a particular structure. They are generated from a single pair  $\{(p[0], \dots, p[b-1]), (p'[0], \dots, p'[b-1])\}$ , where  $p[i]$  and  $p'[i]$  are nibbles. Suppose that the input difference covers the first three nibbles so that  $p[3] = p'[3] = c[3], \dots, p[b-1] = p'[b-1] = c[b-1]$  for some constants  $c[i]$ . Then, the family corresponding to this pair is made by exchanging some nibbles between the two encryptions in the pair so as to obtain the following pairs:

$$\begin{aligned} & \left\{ \begin{array}{l} (p[0], p[1], p[2], c[3], \dots, c[b-1]) \\ (p'[0], p'[1], p'[2], c[3], \dots, c[b-1]) \end{array} \right\} \quad \left\{ \begin{array}{l} (p'[0], p[1], p[2], c[3], \dots, c[b-1]) \\ (p[0], p'[1], p'[2], c[3], \dots, c[b-1]) \end{array} \right\} \\ & \left\{ \begin{array}{l} (p[0], p'[1], p[2], c[3], \dots, c[b-1]) \\ (p'[0], p[1], p'[2], c[3], \dots, c[b-1]) \end{array} \right\} \quad \left\{ \begin{array}{l} (p[0], p[1], p'[2], c[3], \dots, c[b-1]) \\ (p'[0], p'[1], p[2], c[3], \dots, c[b-1]) \end{array} \right\}. \end{aligned}$$

Overall, if there are  $n$  nibbles with nonzero differences in the input, then a family is made of  $2^{n-1}$  pairs and  $2^n$  encryptions.

In the case of PRINCE, we consider differential trails where the input differences are only over one column and such that all the pairs in a family follow the same trail for the first three rounds. For example, the trails we describe in Sect. 4.2 are either followed by all the elements in a family or none of them. A similar heuristic is used in [17] to perform a multiset attack on the SASAS structure.

This behavior comes from the fact that the transition in the trails we study depends only on the transitions occurring during the first round, which are the same in all pairs of a family, and on the actual value of some nibbles to which the difference has not had the time to propagate, which are the same in all encryptions of the structure.

#### 4.2. Our Trails

We consider trails which are completely specified during the first 3 rounds and then propagate with probability 1 for 2.5 rounds before having spread to the full internal state. Figure 6 shows a first trail covering 5.5 rounds in this way which we denote  $\mathcal{T}_1$ . Each array corresponds to the differences between the internal states of two encryptions under 6-round PRINCE, and each cell gives the value of the difference: Light gray corresponds to a fully specified nonzero value at the nibble level (e.g., a difference of 1), dark gray to an unknown nonzero difference and white to a zero difference. A very similar trail with a probability two times smaller,  $\mathcal{T}_2$ , is given in Fig. 14 (see “Appendix C”). To compute their probabilities, we use the difference distribution matrix of the S-Box. If we let the input difference be  $(1, 1, 1, 0, \dots, 0)$ , then  $\mathcal{T}_1$  has a probability of  $2^{-2 \cdot 3} \cdot 2^{-2} \cdot 2^{-2-2-3} = 2^{-15}$  and  $\mathcal{T}_2$  has a probability of  $2^{-2 \cdot 3} \cdot 2^{-2} \cdot 2^{-2-3-3} = 2^{-16}$ .

Querying enough families at random to find one right family for any of these would require  $(2^{-15} + 2^{-16})^{-1} = 2^{14.41}$  families with an input difference over three nibbles, i.e.,  $2^{14.41} \cdot 2^3 = 2^{17.41}$  encryptions. However, we can use structures to decrease this complexity.

We note that the input differences which might lead to an output difference of 1 are those listed in Table 2. As we can see, the second bit from the right in little-endian notation is only involved in  $0 \times 2$  and  $0 \times b$  which, taken together, only have a probability of 1/4 of leading to a difference of 1. Hence, we use the following structures where  $b$  is a bit taking all possible values and  $c$  is a constant across the structure:

bbcb bbcb bbcb cccc cccc ... cccc.

We found experimentally that such structures contain several<sup>1</sup> right families with probability  $2^{-5.9}$  on average when we take into account all possible input differences, i.e.,  $(\delta, \delta', \delta'', 0, \dots, 0)$  where  $\delta, \delta', \delta'' \in \{1, 4, c, d\}$ . Hence, obtaining at least two right families only requires about  $2^{9+5.9} = 2^{14.9}$  queries to the encryption oracle on average.

<sup>1</sup>Actually, a structure of size  $2^{12}$  where the first three nibbles take all values contains 64 right families with probability about  $2^{-5.9}$ . If we reduce these to form the structures of  $2^9$  plaintext/ciphertext encryptions we described, only some of these 64 families are still present, hence the presence of either 0 or several right families in a structure.

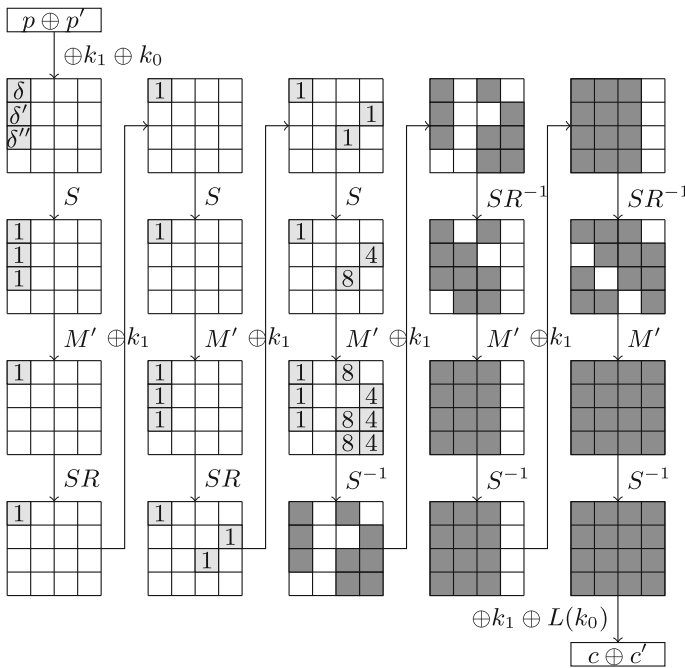


Fig. 6. The 5.5-round trail  $T_1$ .

Table 2. Input differences which might be mapped to a difference of 1 by the S-Box of PRINCE.

Hexadecimal	Binary	Probability
0x1	0001	1/4
0x2	0010	1/8
0x4	0100	1/8
0xb	1011	1/8
0xc	1100	1/4
0xd	1101	1/8

### 4.3. Filtering Right Pairs

Full diffusion has been achieved by the sixth round. Thus, we guess 12 bits to be able to partially invert the last round on one column. More precisely, we first guess the difference in one column before the last MixColumns for one pair. Each guess leads to a candidate for the corresponding column of  $k'_0 \oplus k_1$  that we try against the three other pairs of the family. A guess leads to the correct nibble having a zero difference in every pair of the family with probability  $2^{-3 \cdot 4} = 2^{-12}$ . We repeat this independently over each column and obtain either 64 bits of key material or none at all. Since there are either several right families or none at all in the structures we consider, we only return the key guesses which come from several families as well as the corresponding families.

Note this is a powerful filter: While we expect each family from the structure to yield about one 64-bit candidate, the probability to have a collision is very small.<sup>2</sup>

The same procedure is then used one round earlier to recover 48 bits of  $k_1$ , corresponding to its three first columns. They allow us to partially decrypt the family and check whether differences after the third MixColumns are correct for the three first columns. The probability for a wrong family to pass this test is  $2^{-6 \cdot 4 \cdot 4} = 2^{-96}$ , so we expect to be left with right families only. Finally, the missing 16 bits of  $k_1$  are recovered by performing an exhaustive search.

*Complexity* Querying  $2^{5.9}$  structures of  $2^9$  chosen plaintexts to obtain at least two right families, the time complexity of the key recovery procedure is around  $2^{5.9+9} \cdot 2^{12} \cdot 7$  partial encryptions, which is equivalent to  $2^{29.7} \cdot \frac{4}{6 \cdot 16} \approx 2^{25.1}$  encryptions. The memory complexity is dominated by the storage of the  $2^{14.9}$  required to run this attack.

## 5. Combining Differential Attack with a SAT-Solver

### 5.1. Attacking 4-Round PRINCE with a SAT-Solver

#### 5.1.1. Encoding PRINCE as a CNF Formula

The idea is to generate a CNF formula where a set  $p$  of Boolean variables corresponds to the 64 bits of the plaintext,  $c$  to the 64 bits of the ciphertext and  $k$  to the 128 bits of the key and such that there exists a unique assignment of the variables satisfying the CNF corresponding to the case  $E_k(p) = c$ .

Hence, if we generate such a formula, set the variables in  $p$  and  $k$  to a chosen value and use a SAT-solver to find an assignment satisfying the CNF formula, and the variables in  $c$  will correspond to the ciphertext. Solving such a formula is easy, an observation which we can relate to the fact that the evaluation of a block cipher has to be “easy” for the block cipher to be fast for both encryption and decryption.

Another way to use such a formula is to fix the variables in  $p$  and in  $c$  according to a known plaintext/ciphertext pair, solve the CNF and recover the key from the variables corresponding to it. Unless the number of rounds is very small (at most 3 in the case of PRINCE), solving such a system is impractical. Again, we can relate this observation to the fact that recovering the key given one or several plaintext/ciphertext pair has to be “hard.” Our approach consists in using some knowledge about the internal state of the cipher to simplify the task of the SAT-solver and make such a resolution possible for a higher number of rounds.

In order to encode a PRINCE encryption as a CNF formula, we introduce several sets of 64 Boolean variables corresponding to each step of each round: one for the internal state at the beginning of the round ( $x_r$ ), one for the internal state after going through the S-Box ( $y_r$ ), etc. We also use Boolean variables corresponding to the key bits.

---

<sup>2</sup>Each structure yields  $2^{9-3} = 2^6$  families for each of the  $4^3$  interesting input differences so that we consider the families by groups of  $2^{12}$ . This implies that a collision has a probability of about  $\binom{2^{12}}{2} \cdot 2^{-64} \approx 2^{-41}$ .

Our task is then to create a CNF formula connecting these variables in such a way as to ensure that, for instance if  $k[0, \dots, 63]$  is fixed, it has only one solution where  $y_r[0, \dots, 63]$  is indeed the image of  $x[0, \dots, 63]_r$  by  $S$ , etc.

In order to encode the linear layer, we use the alternative representation of  $M'$  from [10] where it was shown that  $M'$  operates on columns of four bits independently by first rotating them by a column-dependent number of bit and then XOR-ing the parity of the column in each bit. We thus add variables corresponding to the hamming weights of the columns and encode the corresponding XOR's as CNF formulas. The  $SR$  operation is only a permutation of the bits, so we simply set the corresponding bits to be equal.

The encoding of the S-Box is less simple to obtain. In order to find the best one, we chose to look for it directly instead of using the ANF as an intermediate step. Indeed, since the S-Box is  $4 \times 4$ , it is small enough for us to brute force all clauses<sup>3</sup> involving input and output bits and check whether they hold for every input.

Doing this leads us to find 29 clauses with three variables. However, they are not sufficient to completely specify the S-Box, so we used a greedy algorithm to find the best clauses with four variables to add to this encoding. In the end, we have 29 clauses with three variables and nine clauses with four variables which are such that the only solutions of the CNF made of all these clauses are all the assignments corresponding to pairs  $(x, S(x))$  for all  $x \in [0, 15]$ .

These clauses with three variables can be interpreted as simple implications. For example, if  $o[3, \dots, 0]_b = S(i[3, \dots, 0]_b)$ , then the following two clauses hold with probability 1:

$$(i[1]_b \vee o[2]_b \vee o[3]_b) \wedge (i[1]_b \vee o[1]_b \vee o[2]_b).$$

They are logically equivalent to the following implication:

$$\overline{i[1]_b} \implies ((o[2]_b \vee o[3]_b) \wedge (o[1]_b \vee o[2]_b)).$$

### 5.1.2. Differential Over-Definition

The approach consisting in using the knowledge from a differential trail to ease the task of a SAT-solver used to attack a cryptographic primitive has been explored in [18] in order to attack MD4 and MD5. The authors of this paper first use heuristic methods to find a high probability differential trail leading to a collision and then use a SAT-solver to find a pair of messages which satisfies this trail. In the same paper, we can find the following observation:

An interesting result of our experiments with SAT solvers is the importance of having a differential path encoded in the formula.

As we shall see, this also holds for block ciphers. Attacking 4- round PRINCE-core takes more than 10 hours if we simply encode as a CNF that some plaintexts are encrypted into known ciphertexts, but we can both drastically reduce this time while breaking PRINCE with its whitening keys using *differential over-definition*.

---

<sup>3</sup>A clause is the logical OR of several variables, e.g.,  $a \vee b$ ,  $a, \bar{a} \vee b \vee \bar{c}$  where  $\bar{x}$  is the negation of  $x$ .

**Definition 2.** We call *differential over-definition* (or DOD) the following algorithm which simplifies a CNF formula knowing that the variables correspond to bits of the internal state of an encryption following a certain trail.

For all pairs of variables in the CNF, proceed as follows:

- If they are assumed to be equal, replace all occurrences of the first one by the second one.
- If they are assumed to be different, replace all occurrences of the first one by the negation of the second one.

While the idea behind this algorithm is simple, it is necessary for cryptographers to implement it efficiently “by hand.” Indeed, the only input of a SAT-solver is a CNF formula, i.e., merely a list of clauses from which deriving what variables are equal to each other without knowledge of the structure of the problem is far from trivial. For instance, it would be necessary for the SAT-solver to “understand” that the set of clauses used to model one S-Box call all correspond to a unique function so that identical inputs lead to identical outputs, all this without having any distinction between the input and output bits. That is why differential over-definition, an easy algorithm for the cryptographer to implement, is a valuable preprocessing step when using a SAT-solver for cryptography leading to gains in time complexity of several orders of magnitude.

This algorithm can be implemented efficiently using a hash table containing the correspondences between the variables. Once this algorithm has been run, the CNF is over-defined: The solution would have been such that the equalities hold anyway, but there are less variables and less clauses in the CNF. However, if the pair actually does not follow the trail, the CNF has become unsatisfiable. This is a difference between our work and the one described in [18]: We do not always know before hand whether the CNF has a solution. We can think of this as a trade-off between “solving one CNF known to be true” and “solving many over-defined CNF’s which may or may not be true”: The second approach loses time by requiring several calls to a SAT-solver, but these calls take less time thanks to the over-definition.

Such an over-definition can be used in different ways.

1. Propagating only the zero differences holding with probability 1 inside a group of eight encryptions with many zero differences is enough to reduce the time complexity of an attack on four rounds from more than 10 hours to a few seconds (see below). Furthermore, such a formula is always true.
2. Instead of implementing an algorithm recovering the key from a pair following a particular trail by peeling of layer after layer of encryption in our attack on six rounds described in the remainder of this section, we simply reused the code of our attack on four rounds and over-defined the CNF modeling the encryptions of right pairs according to the high probability trail we used.

We implemented the attack described in Algorithm 1 to attack 4-round PRINCE (with its whitening keys) using the SAT-solver Minisat [19] and obtained an average total time of 5.13s and average time spent solving the CNF of 3.06s. The designers of PRINCE did not consider SAT-based attacks, but they did investigate algebraic attacks. They manage to attack 4-round PRINCE-core in less than 2s, while our attack requires about 5s to attack 4-round PRINCE, a cipher which uses twice as much key material.



**Algorithm 1** Using differential over-definition to enable an attack on 4-round PRINCE.

---

Query  $2^{10}$  plaintexts/ciphertexts where the first 10 bits take all possible values.  
 Select a subset of eight plaintexts/ciphertexts maximizing the number of zero differences in the output.  
 Encode the eight encryptions as a CNF  $A$ .  
 Overdefine  $A$  by propagating zero differences with probability 1.  
 Use a SAT-solver to retrieve the key bits from  $A$   
**return**  $k_0||k_1$

---

## 5.2. Differential Attacks on 6-Round PRINCE

Another possible use for a SAT-solver is the handling of the tedious details of an actual attack implementation, e.g., the finding of an efficient guessing strategy. We illustrate this by performing the recovery of the second chunk of the master key in our 6-round differential attack using a SAT-solver. Pseudo-code describing this attack on 6-round PRINCE is provided in Algorithm 2.

**Algorithm 2** Using trails  $\mathcal{T}_1, \mathcal{T}_2$  and a SAT-solver to recover the complete key  $k_0||k_1$  of 6-round PRINCE.

---

**while** the key has not been retrieved **do**  
 Query a structure  $\mathcal{S} = ((p^0, c^0), \dots, (p^{2^{12}-1}, c^{2^{12}-1}))$   
 $H \leftarrow$  empty hash table of lists of families indexed by 64-bit integers  
**for all** families  $\mathcal{F}$  in  $\mathcal{S}$  **do**  
**for** one pair in  $\mathcal{F}$  **do**  
**for all** columns of the internal state **do**  
**for all** the  $2^{12}$  possible differences **do**  
 Deduce the corresponding 16-bit key material  $k_{16}$   
**for all** remaining pairs in  $\mathcal{F}$  **do**  
 Invert key addition for the column using  $k_{16}$   
 Invert  $S^{-1}$  for the column  
 Invert  $M'$  for the column  
**end for**  
**if** correct nibble has zero difference in all pairs **then** store  $k_{16}$   
**end for**  
**end for**  
**end for**  
 Combine all guesses from each column into 64-bit guesses  
**for all** 64-bit guesses  $k_{64}$  append  $\mathcal{F}$  to  $H[k_{64}]$   
**end for**  
**for all**  $k_{64}$  among the keys of  $H$  **do**  
**if**  $H[k_{64}]$  contains strictly more than one element **then**  
**for all** families  $\mathcal{F}$  in  $H[k_{64}]$  **do**  
 Generate a CNF  $A$  encoding all encryptions in  $\mathcal{F}$  with same key such that  $k_1 + L(k_0) = k_{64}$ .  
**for all** trails  $\mathcal{T}$  in  $\{\mathcal{T}_1, \mathcal{T}_2\}$  **do**  
 $B \leftarrow DoD(A, \mathcal{T})$   
**if**  $B$  is satisfiable **then** retrieve  $k_0||k_1$  from the solution of  $B$  and return it  
**end for**  
**end for**  
**end if**  
**end for**  
**end while**

---

We ran this attack ten times and found that about  $2^{5.75}$  structures were needed on average. The filtering step is the most time-consuming: Finding a right pair requires about 6 minutes, while the SAT-solver requires about 0.5s to recover the full key or (rarely) to discard the pair. For this reason, we approximate the complexity of this attack by the complexity of its filtering step. Memory complexity is dominated by the SAT-solver but is (well) below 1 GB, i.e., (well) below  $2^{27}$  64-bit blocks.

## 6. Structural Analysis of PRINCE

The  $\alpha$ -reflection introduced along with PRINCE [5] is the name given to the following property of a block cipher  $E_k$ :  $E_k^{-1} = E_{k \oplus \alpha}$ . In other words, there is a constant  $\alpha$  such that decryption for a key  $k$  is the same operation as encryption under key  $k \oplus \alpha$ . PRINCE-core implements this property by having a three-part structure as described here:

$$E_{k_1}^c = F_{k_1 \oplus \alpha}^{-1} \circ I \circ F_{k_1},$$

where  $F_k$  corresponds to five rounds of a classical substitution–permutation network construction and where  $I$  is an involution.

Since we are going to study the structure of the cycles of different functions in a fashion similar to the way Biryukov analyzed the inner rounds of some involutorial ciphers in [20], we define the *cycle type* of a permutation.

**Definition 3.** The cycle type of a permutation  $\pi$  is an (ordered) multiset containing the cycle lengths of the permutation. The cycle type of  $\pi$  is denoted by  $\mathcal{L}(\pi)$ .

For instance, for  $\pi = (0\ 1\ 2)(3\ 4)(5\ 6)$  we have  $\mathcal{L}(\pi) = \{2, 2, 3\}$ , while for  $\pi = (0\ 1\ 2)(3\ 4\ 5)(6)$ ,  $\mathcal{L}(\pi) = \{1, 3, 3\}$ . In what follows, we do not represent the round constants for the sake of simplicity. However, not only do our result hold in their presence but we could actually generalize them to any key schedule preserving the fact that the subkeys of symmetric rounds have a XOR equal to  $\alpha$ .

### 6.1. Small Cycles in Round-Reduced PRINCE

The central involution is  $I = S^{-1} \circ M' \circ S$ . Therefore, it is isomorphic to  $M'$ , a linear involution operating on each column of the internal state independently. It is easy to check experimentally the result given in [7] stating that  $M'$  has exactly  $2^{32}$  fixed points, meaning that  $I$  also has  $2^{32}$  fixed points. Therefore,  $I$  has  $2^{32}$  cycles of length 1 and  $2^{63} - 2^{31}$  cycles of length 2.

The cycle type of  $I^\alpha : x \mapsto I(x) \oplus \alpha$  is more sophisticated but still contains a fair amount of small cycles. After noting that both  $I$  and  $x \mapsto x \oplus \alpha$  operate on each column of the internal space independently, we denote  $I_i^\alpha$  the restriction of  $x \mapsto I(x) \oplus \alpha$  to column  $i$  and  $I_i$  that of  $I$ . Since each of the  $I_i^\alpha$ 's operates only on a space of size  $2^{16}$ , it is easy to generate their complete cycle structures independently by searching the whole space. Each  $I_i^\alpha$  has a cycle type made of many “small” cycles, the largest having a length of 2844. This is explained by the fact that both  $I$  and  $x \mapsto x \oplus \alpha$  are involutions and

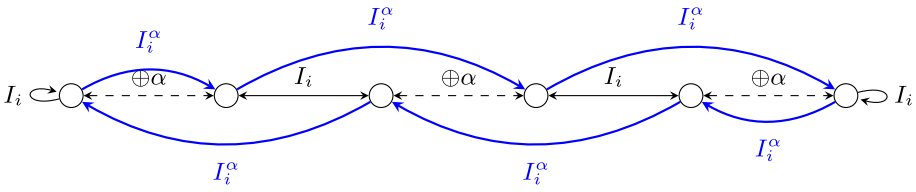


Fig. 7. The structure of a cycle of  $I_i^\alpha$  for  $i \in [0, 3]$ .

Table 3. Information about the cycle type of  $I^\alpha$ , where  $\ell(x)$  is the length of the cycle on which  $x$  is.

Cycle length $\ell$	#[cycles of length $\ell$ ]	$P[\ell(x) = \ell, x \text{ drawn uniformly}]$
1	0	0
2	$2^7$	$2^{-57}$
4	$2^{10.25}$	$2^{-53.75}$
8	$2^{15.46}$	$2^{-48.54}$
10080	$2^{33.06}$	$2^{-17.63}$
110880	$2^{31.96}$	$2^{-15.27}$
$\leq 2^{10}$	–	$2^{-22.4}$
$\leq 2^{15}$	–	$2^{-12.4}$
$\leq 2^{24}$	–	$2^{-4.1}$

each column of  $I$  has exactly  $2^8$  fixed points. Thus, most of the cycles have a particular structure described in [21] which we recall in Fig. 7. We remark that to each cycle of  $I_i^\alpha$  correspond two fixed points of  $I_i$ .

After generating the cycle type for each  $I_i^\alpha$ , we combine them to obtain the cycle type of  $x \mapsto I(x) \oplus \alpha$  using Algorithm 3. The cycle type of this function is too complex to be printed completely, but some information extracted from it is given in Table 3. If we pick  $x$  uniformly at random, the expected length of the cycle it is on is  $2^{30.7}$ .

**Algorithm 3** Generating the cycle type of  $I^\alpha$  from those of its columns.

```

for  $i \in [0, 3]$  do
     $\mathcal{L}_i \leftarrow$  List of the cycle length of  $I_i^\alpha$ 
end for
 $\mathcal{L} \leftarrow$  Hash table indexed by integers
for  $(\ell_0, \ell_1, \ell_2, \ell_3) \in \mathcal{L}_0 \times \mathcal{L}_1 \times \mathcal{L}_2 \times \mathcal{L}_3$  do
     $\ell \leftarrow \text{lcm}(\ell_0, \ell_1, \ell_2, \ell_3)$ 
     $\mathcal{L}[\ell] \leftarrow \mathcal{L}[\ell] + \ell^{-1} \cdot \prod_{i=0}^3 \ell_i$ 
end for
return  $\mathcal{L}$ 
    
```

Recall that  $E_{k_1}^{c,4}$  is the permutation of  $\{0, 1\}^{64}$  corresponding to an encryption under key  $k_1$  by PRINCE-core reduced to four rounds. Then,  $x \mapsto E_{k_1}^{c,4}(x) \oplus \alpha$  has the same cycle type as  $I^\alpha$  due to the cancellation of the last round of one encryption with the first round of the next. Indeed, to each cycle of this function corresponds one of  $I^\alpha$ , as

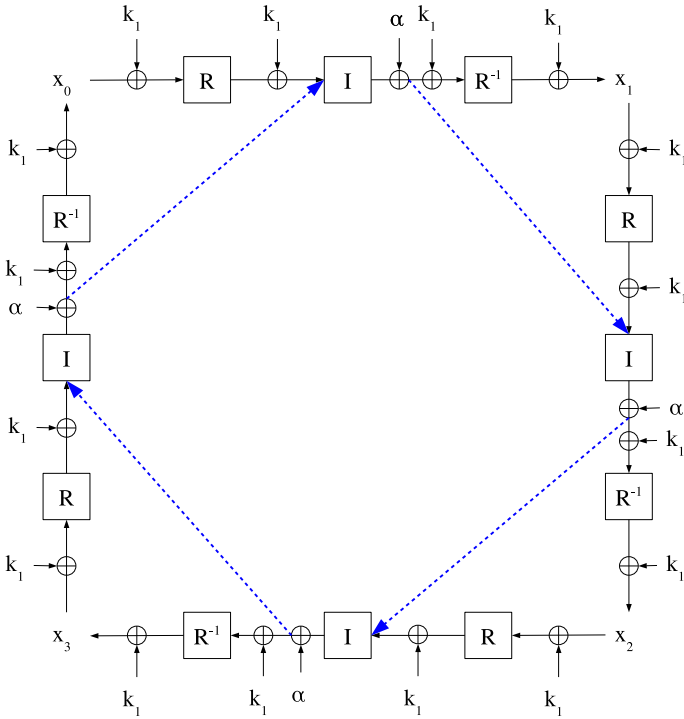


Fig. 8. Correspondence between a cycle of  $x \mapsto E_{k_1}^{c,4}(x) \oplus \alpha$  and a cycle of  $I^\alpha$ .

illustrated in Fig. 8 where a cycle  $(x_0, x_1, x_2, x_3)$  of length 4 of  $x \mapsto E_{k_1}^{c,4}$  is represented along with the corresponding cycle of  $I^\alpha$  (dashed line).

A first consequence of these observations is the existence of a distinguisher for 4-round PRINCE-core requiring about  $2^{27.4}$  adaptively chosen plaintexts. As stated in Table 2, an element picked at random is on a cycle of length at most  $2^{15}$  with probability  $2^{-12.4}$ . Therefore, we repeat multiple times the experiment consisting in picking an element  $x$  uniformly at random and then check whether it is on a cycle of length at most  $2^{15}$  by iterating  $x \mapsto E_{k_1}^{c,4}(x) \oplus \alpha$  at most  $2^{15}$  times. The experiment is a success if  $x$  is on a cycle of length at most  $2^{15}$ . If the permutation is  $E_{k_1}^{c,4}$  for some  $k_1$ , then its probability of success is  $2^{-12.4}$ , but if the permutation is a random permutation,<sup>4</sup> then the probability of success becomes  $2^{-49}$ . We confirmed experimentally the success probability of this experiment for  $E_{k_1}^{c,4}$ .

A second consequence is the existence of “small” sets of plaintext/ciphertext encryptions where the set of the ciphertexts is the image of the set of the encryptions by a

<sup>4</sup>Recall that the probability for  $x$  to be on a cycle of length  $\ell$  for a permutation of  $[0, n - 1]$  is  $1/n$ . Indeed, let  $x_0 = x$ , we require that  $x_1 = p(x_0) \notin \{x_0\}$ ,  $x_2 = p(x_1) \notin \{x_0, x_1\}$ , ... so the probability that  $x$  belongs to an  $\ell$ -cycle is  $\frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdots \frac{n-\ell}{n-\ell+1} \cdot \frac{1}{n-\ell} = 1/n$ . Hence, the probability that the length is smaller than  $2^{15}$  for a permutation of  $[0, 2^{64} - 1]$  is  $\sum_{\ell=1}^{2^{15}} 2^{-64} = 2^{-49}$ .

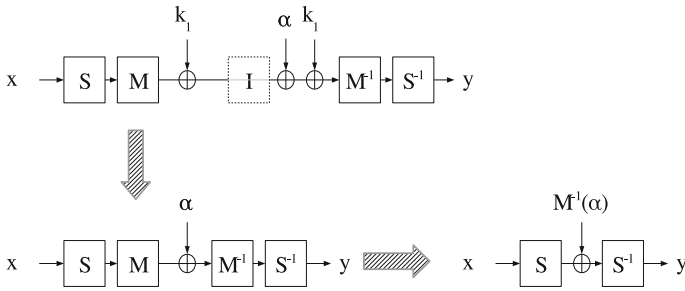


Fig. 9. Simplification of the four center rounds if the input of  $I$  is a fixed point.

function significantly simpler than a PRINCE encryption. This topic is studied in the next section.

### 6.2. Simplifications of PRINCE’s Representation

The particular cycle types of the round-reduced versions of PRINCE studied above lead to simpler alternative representations of the encryption algorithm.

#### 6.2.1. Consequences of the Cycle Type of $I$

Suppose that an encryption is such that the input of  $I$  is one of the  $2^{32}$  fixed points of this function. Then, the key addition before and after this function cancels each other so that only the addition of  $\alpha$  remains. Then, since  $M$  is linear, the operations  $M^{-1} \circ (\oplus \alpha) \circ M$  become simply the addition of  $M^{-1}(\alpha)$ . Thus, the four center rounds— minus the first and last key addition—become a simple S-Box layer which we denote  $S'$  and which is defined by

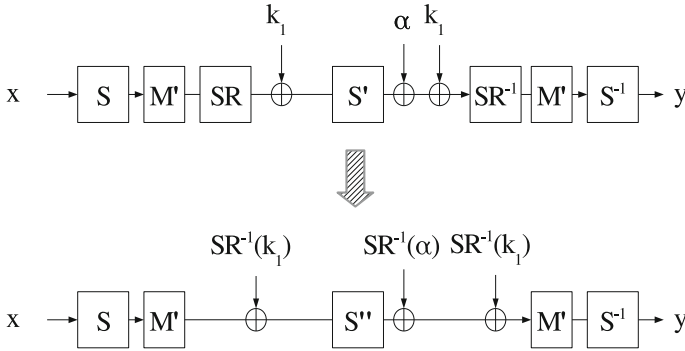
$$S'(x) = S^{-1}(S(x) \oplus M^{-1}(\alpha)).$$

This simplifying process is summarized in Fig. 9. Note that if  $M^{-1}(\alpha)$  has any nibble equal to 0, then the function  $S'$  is the identity for this nibble. However, for the value of  $\alpha$  chosen by the designers of PRINCE, there is no such nibble.

The simplification goes further. Indeed, since  $S'$  operates only at the nibble level, it commutes with the operations  $SR$  and  $SR^{-1}$  (up to a reordering of the S-Boxes in  $S'$ ). Therefore, if we add one round before and one round after  $S'$ , we can replace  $SR^{-1} \circ S' \circ SR$  by  $S''$  where  $S''$  is another S-Box layer. Hence, six rounds of PRINCE operate on each column of the internal state independently: Each output bit depends only on 16 bits of the input, 28 bits<sup>5</sup> of  $k_1$  and at most 18 bits of  $k_0$ . This simplification is summarized in Fig. 10.

Similar simplifications occur if instead of having a fixed point we have a particular collision between two encryptions. This setting corresponds to the so-called *mirror slide attack* described by Dunkelman et al. in [22]. Consider two encryptions  $(p^0, c^0)$  and

<sup>5</sup>In each column, 16 bits from the corresponding column of  $k_1$  are used as well as 16 bits from the corresponding column of  $SR^{-1}(k_1)$ . Since the top nibble of these two sets is the same, we are left with  $32 - 4 = 28$  bits.



**Fig. 10.** Simplification of the six center rounds if the input of  $I$  is a fixed point.

$(p^1, c^1)$  by PRINCE-core as follows:

$$c^0 = E_{k_1}^c(p^0) = (F_{k_1 \oplus \alpha}^{-1} \circ I \circ F_{k_1})(p^0)$$

$$c^1 = E_{k_1}^c(p^1) = (F_{k_1 \oplus \alpha}^{-1} \circ I \circ F_{k_1})(p^1)$$

which are such that  $F_{k_1}(p^0) = I(F_{k_1}(p^1))$ . In this case, we have that

$$c^0 = (F_{k_1 \oplus \alpha}^{-1} \circ F_{k_1})(p^1)$$

$$c^1 = (F_{k_1 \oplus \alpha}^{-1} \circ F_{k_1})(p^0),$$

where six rounds of  $F_{k_1 \oplus \alpha}^{-1} \circ F_{k_1}$  can be simplified exactly as described and therefore only operate on each column separately.

In conclusion, if an encryption is such that the input of  $I$  is a fixed point of this function or if two encryptions form a mirror slide pair, then four rounds of PRINCE consist simply in 16 parallel operations on each nibble and six rounds of PRINCE in four parallel operations on each column.

### 6.2.2. Consequences of the Cycle Type of $I^\alpha$

Consider a sequence of plaintexts  $(p^0, \dots, p^{\ell-1})$  and their corresponding ciphertexts  $(c^0, \dots, c^{\ell-1})$  such that the input  $x_5^i \oplus k_1$  of the sixth round for the plaintext  $p^i$  is the image of  $x_5^{i-1} \oplus k_1$  by  $I^\alpha$ . We call such a sequence a *cycle set*, and we give a representation of such a sequence on Fig. 11: If two values are equal, then they are connected by a line; red lines correspond to the cycle of  $I^\alpha$  this set is built out of and blue lines correspond to the propagation of these equalities through identical operations, namely  $x \mapsto k_1 \oplus R^{-1}(x \oplus k_1)$ .

There is a unique function mapping  $p^i$  to  $c^{i-1}$  in every cycle set which corresponds to the encryption algorithm where the four center rounds have been removed and replaced by a simple addition of  $\alpha$ . This means that this function undergoes the simplifications described above except that these cover two more rounds. In particular, for 6-round PRINCE-core, the function mapping  $p^i$  to  $c^{i-1}$  only operates at the nibble level and, for

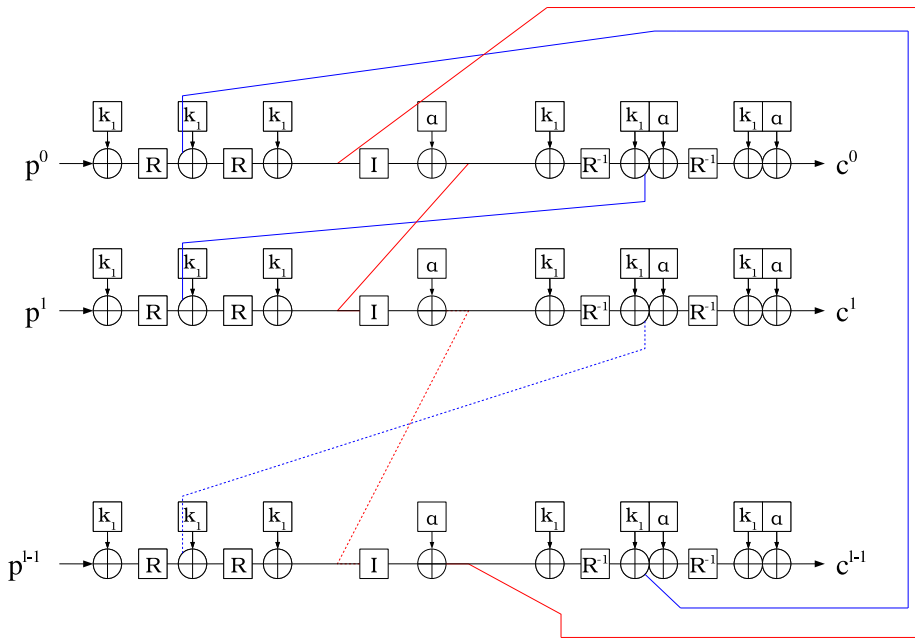


Fig. 11. A cycle set of 6-round PRINCE-core.

8-round PRINCE-core, it operates at the column level. At least 10 rounds are necessary to obtain full diffusion out of the 12 PRINCE has.

The cycle sets we consider cover the four center rounds of PRINCE, but it is possible to generalize this construction to an arbitrary amount of rounds. However, the cycle set sizes are abnormally small in this case because of the cycle type of  $I^\alpha$ . Indeed, a random plaintext/ciphertext pair is in a cycle set of size  $2^{30.7}$  and in a cycle set of size smaller than  $2^{15}$  with probability  $2^{-12.4}$ . In other cases, including *a priori* if we have a cycle covering at least six rounds, the expected size of a cycle set is the expected size of the cycle of a random permutation a random element is on, namely  $2^{63}$ .

Should the cycle sets of PRINCE become identifiable, the security of up to eight rounds may be compromised as the alternative versions of the cipher we described in this section are much weaker than the original cipher. Furthermore, since small cycles are not unlikely to be found, the data complexity of such an attack may remain feasible.

### 7. Conclusion

We looked for practical attacks which would hinder the security provided by round-reduced versions of PRINCE in a realistic framework provided by the designers of this cipher. We found that approaches based on a meet-in-the-middle, SAT-based or, surprisingly, differential framework can all lead to practical attacks on up to half of the rounds. We checked our results by actually implementing one of our attacks. As a matter

of fact, our attacks were the best submitted to the PRINCE challenge for four, six and eight rounds. Furthermore, during our investigations on PRINCE we discovered a new attack on ten rounds which despite its data complexity of  $2^{57}$  chosen plaintexts has a reasonable complexity and a very (very!) motivated adversary could run it.

We also identified some simplifications of the encryption occurring because of the small cycles of the inner rounds of this block cipher, thus shedding new light on the consequences of the  $\alpha$ -reflection as it is implemented in PRINCE.

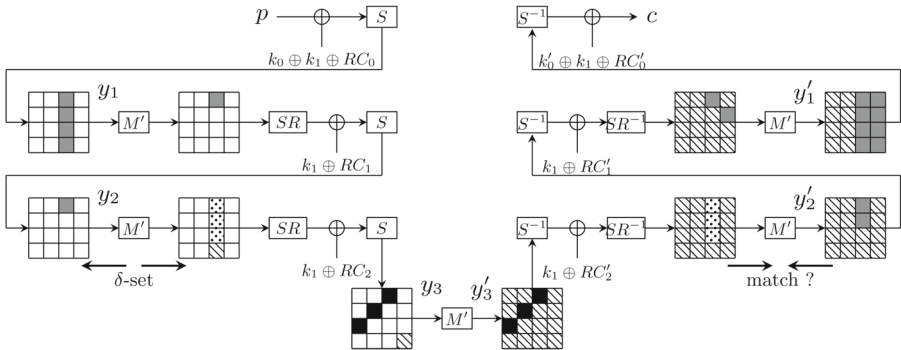
### Acknowledgements

The authors thank Alex Biryukov for useful discussions about the differential attack on PRINCE. We also thank NXP Semiconductors for organizing the PRINCE challenge and sending us our rewards! The work of the authors was supported by the CORE ACRYPT project (ID C12-15-400992) and funded by the *Fonds National de la Recherche* (Luxembourg).

### A The Second 6-Round Attack

See Fig. 12.

#### A The Second 6-round Attack



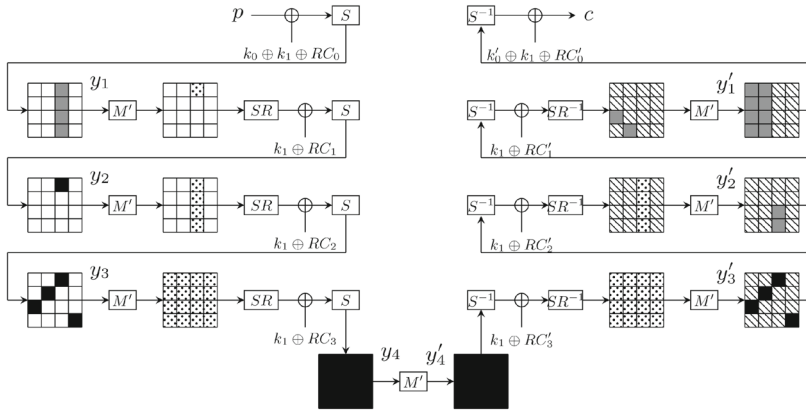
**Fig. 12.** 6r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.



### B The Second 8-Round Attack

See Fig. 13.

#### B The Second 8-round Attack

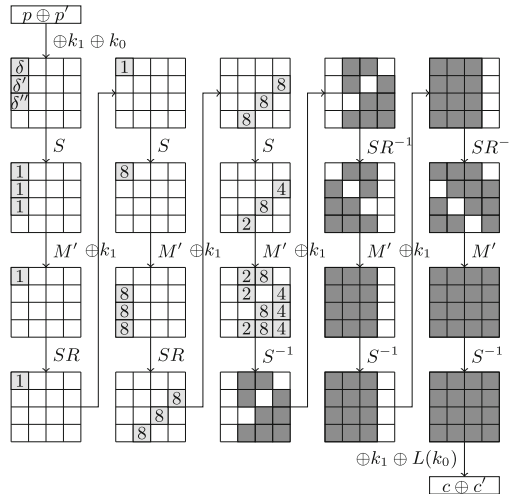


**Fig. 13.** 8r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

### C The Second 5.5-Round Trail

See Fig. 14.

#### C The Second 5.5 Rounds Trail



**Fig. 14.** The second 5.5-round trail  $\mathcal{T}_2$ .

### D Simple Meet-in-the-Middle Attacks

In this section, we describe two attacks on round-reduced PRINCE. Both are simple meet-in-the-middle attacks requiring only few known plaintext/ciphertext pairs to work.

#### D.1 4-Round Attack

##### Simple Attack

We begin by presenting a simple attack on 4-round PRINCE with a complexity around  $2^{40}$  which is depicted in Fig. 15. It is based on the two following equations involving few bits of the middle states  $y$  and  $y'$ :

$$\begin{cases} y[38]_b \oplus y[46]_b = y'[38]_b \oplus y'[46]_b \\ y[39]_b \oplus y[43]_b \oplus y[47]_b = y'[47]_b \end{cases}$$

Let  $K_p$  (resp.  $K_c$ ) be the key bits required to compute  $y[38]_b \oplus y[46]_b$  and  $y[39]_b \oplus y[43]_b \oplus y[47]_b$  from  $p$  (resp.  $y'[38]_b \oplus y'[46]_b$  and  $y'[47]_b$  from  $c$ ). Then, the attacks scenario is:

1. Ask for  $n$  known plaintext/ciphertext pairs.
2. Let  $T$  be an empty hash table.
3. For all possible values of  $K_p$ , do
  - (a) for  $j$  from 1 to  $n$  compute  $y^j[38]_b \oplus y^j[46]_b$  and  $y^j[39]_b \oplus y^j[43]_b \oplus y^j[47]_b$  from the  $j$ -th plaintext
  - (b) make the sequence  $s = [y^1[38]_b \oplus y^1[46]_b, y^1[39]_b \oplus y^1[43]_b \oplus y^1[47]_b, \dots]$
  - (c) add the value of  $K_p$  to  $T[s]_b$
4. For all possible values of  $K_c$ , do
  - (a) for  $j$  from 1 to  $n$  compute  $y'^j[38]_b \oplus y'^j[46]_b$  and  $y'^j[47]_b$  from the  $j$ -th ciphertext
  - (b) make the sequence  $s = [y'^1[38]_b \oplus y'^1[46]_b, y'^1[47]_b, \dots]$

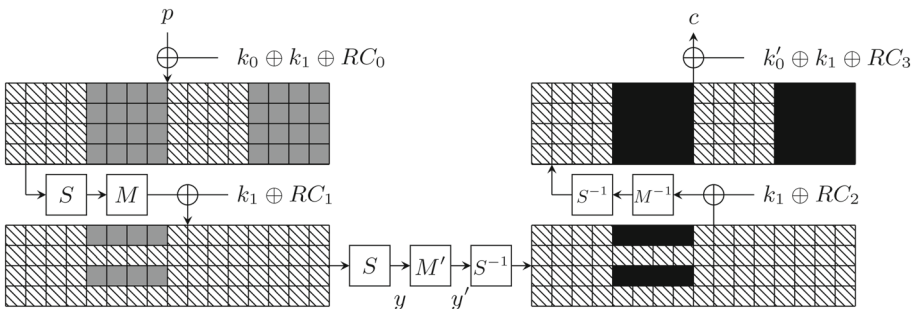


Fig. 15. 4r attack. Hatched bits play no role .

- (c) check whether  $T[s]$  is empty or not. If  $T[s]$  is empty, then the guess of  $K_c$  is wrong. Otherwise,  $T[s]$  contains the possible value(s) for  $K_p$  and, if  $n$  is large enough, this will happen only for the right guess.

In our case, both  $K_p$  and  $K_c$  can assume only  $2^{40}$  values and thus 40 plaintext/ciphertext pairs are enough to get only one candidate for  $K_p \cup K_c$  with high probability. Thus, the complexity of this attack is 40 known plaintexts and around  $2^{40}$  for both time and memory.

*Saving data and memory* First, we stress that  $K_p$  (resp.  $K_c$ ) can be safely replaced by any basis of the vector space spanned by itself, so let consider it as a vector space over  $\mathbb{F}_2$ . Now, we are interested by the vector space  $K_p \cap K_c$ . Here, a basis of this vector space is:

$$\{k_1[36..39]_b, k_1[44..47]_b, k_0[37..39]_b, k_0[45..47]_b, k_0[40]_b \oplus \dots \\ \oplus k_0[44]_b \oplus k_1[40]_b \oplus \dots \oplus k_1[43]_b\}.$$

Thus, only 33 plaintext/ciphertext pairs are needed to discard the wrong guesses and the attack scenario becomes:

1. Ask for  $n$  known plaintext/ciphertext pairs.
2. For all possible values of  $K_p \cap K_c$ , do
  - (a) Let  $T$  be an empty hash table.
  - (b) Partially encrypt/decrypt the plaintext/ciphertext pairs.
  - (c) For all possible values of  $K_p$ , do
    - (i) for  $j$  from 1 to  $n$  compute  $y^j[38]_b \oplus y^j[46]_b$  and  $y^j[39]_b \oplus y^j[43]_b \oplus y^j[47]_b$  from the  $j$ -th plaintext
    - (ii) make the sequence  $s = [y^1[38]_b \oplus y^1[46]_b, y^1[39]_b \oplus y^1[43]_b \oplus y^1[47]_b, \dots]$
    - (iii) add the value of  $K_p$  to  $T[s]$
  - (d) For all possible values of  $K_c$ , do
    - (i) for  $j$  from 1 to  $n$  compute  $y^j[38]_b \oplus y^j[46]_b$  and  $y^j[47]_b$  from the  $j$ -th ciphertext
    - (ii) make the sequence  $s = [y^1[38]_b \oplus y^1[46]_b, y^1[47]_b, \dots]$
    - (iii) check whether  $T[s]$  is empty or not. If  $T[s]$  is empty, then the guess of  $K_c$  is wrong. Otherwise,  $T[s]$  contains the possible value(s) for  $K_p$  and, if  $n$  is large enough, this will happen only for the right guess.

All in all the memory requirement is approximately  $25 \times 2^{25} \times 2^{-3} \approx 2^{26.7}$  bytes and the time complexity around  $33 \times 2 \times 2^{40} \times 40 / (4 \times 64) \approx 2^{43.4}$  encryptions.

*Key recovery* At the end of the attack, we know (or have very few candidates for)  $K_p \cup K_c$ . But we still need  $128 - 65 = 63$  key bits to fully recover  $k_0$  and  $k_1$ . Performing an exhaustive search at this point would increase the overall complexity of the attack and thus is not a good idea. Instead, it is better to perform another meet-in-the-middle attacks successively. For instance, the attack depicted in Fig. 16 allows to recover 20 more key bits with a time complexity around  $2^{20}$ .

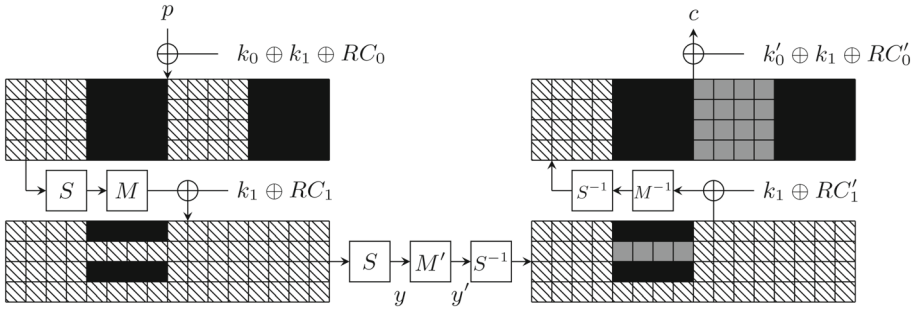


Fig. 16. 4r attack. Black bits are known. Hatched bits play no role.

6-Round Attack

The 4-round attack can be extended to a 6-round attack as depicted in Fig. 17. Now, the dimension of both  $K_p$  and  $K_c$  is 96 and the dimension of the intersection is 64. Thus, it leads to an attack requiring  $(2 \times 96 - 64)/2 = 64$  known plaintext/ciphertext pairs, with a memory complexity around  $(96 - 64) \times 2^{96-64} \times 2^{-3} = 2^{34}$  bytes and a time complexity of approximately  $2 \times 64 \times 2^{96} \times 104/(6 \times 64) \approx 2^{101.1}$  encryptions.

Finally, an exhaustive search can be performed to retrieve  $k_0$  and  $k_1$  without increasing the overall complexity.

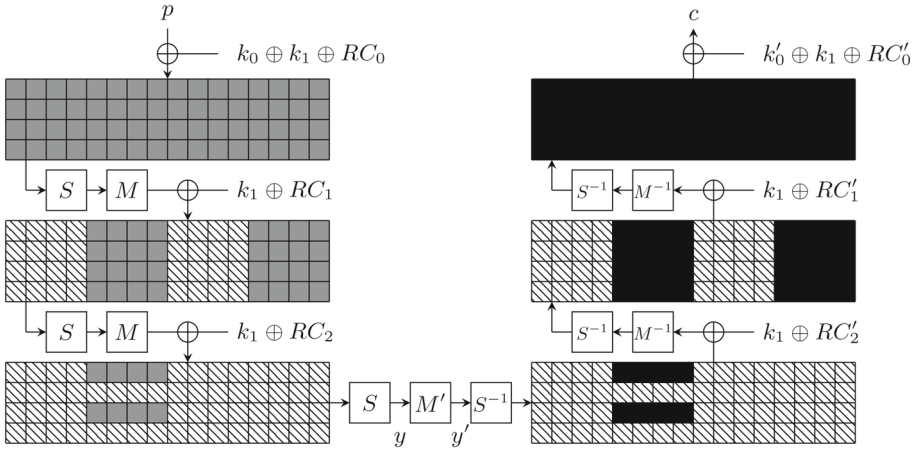


Fig. 17. 6r attack. Hatched bits play no role.

### E Improved 8-Round Attack

In this section, we concisely describe a second attack against 8-round PRINCE, still based on Demirci and Selçuk technique, requiring much less memory than the attack described in Sect. 3.2.

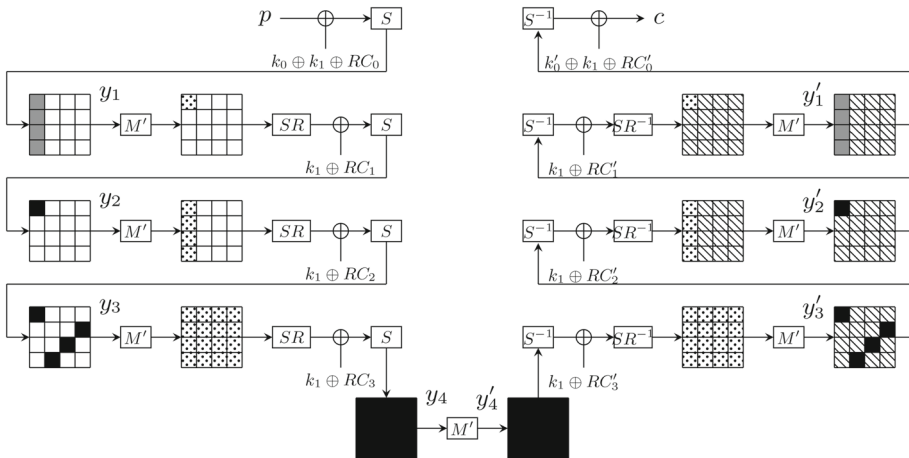
#### Step 1

The first step of the attack is depicted in Fig. 18. This is a classical Demirci and Selçuk attack against 8-round PRINCE. The nibble requiblack in the online phase is in gray and can take  $2^{8 \times 4} = 2^{32}$  values. In another hand, nibbles requiblack in the offline phase are in black and can assume  $2^{80}$  values thanks to the (lack of) key schedule.

Here,  $\delta$ -sets contain  $2^4 = 16$  messages and the check is performed on sequences of  $(16 - 1) = 15$  differences in one nibble, *i.e.*, on 60-bit sequences. As more than  $2^{60}$  sequences are computed during the offline phase, the attack does not filter values of gray nibbles.

#### Step 2

The idea is to switch the online and the offline phases (and actually they are now both performed online). Given a plaintext/ciphertext pair  $(P, C)$ , we compute the  $2^{32}$  possible 60-bit sequences and store them in a hash table. Then for each value of the black nibbles, we compute the corresponding 60-bit sequence and check whether it belongs to the hash table. Only  $2^{80-60+32} = 2^{52}$  values should pass this test.



**Fig. 18.** 8r attack. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

Step 3

We notice that, given a structure of  $2^{16}$  plaintexts such that the first column is active, while the other ones are constant, each column of state  $y_3$  takes all the possible values too. Hence, we can fix the value to 0 of four black nibbles of  $y_4$  (shiftrows of a column), decreasing the time complexity by a factor  $2^{16}$ . As a result, the hash table now has to contain  $2^{48}$  sequences,  $2^{32}$  for each of the  $2^{16}$  plaintext/ciphertext pairs.

Step 4

To decrease further the number of possible values for the black nibbles, we apply the same attack, but we change the nibble on which the check is performed as shown in Fig. 19. So we have to store another table containing  $2^{48}$  60-bit sequences and guess the value of the black nibble of  $y'_2$ . Only  $2^{52+4-60+48} = 2^{44}$  values should pass this test.

Step 5

As in step 4, we perform the two other attacks corresponding to the two other positions for the black nibble on  $y'_2$ . Only  $2^{28}$  values should pass this test.

Step 6

We switch again online and offline phases, coming back to a classical Demirci and Selçuk attack. We run successively the four previous attacks to retrieve a unique plaintext/ciphertext pair together with a unique value for the first column of  $y_1$  and the whole state  $y'_1$ . Indeed, we did not check whether the plaintext/ciphertext pair and the first column of  $y_1$  are the same for the four attacks, while they have to. Thus, only the right ones should remain at the end of the attack.

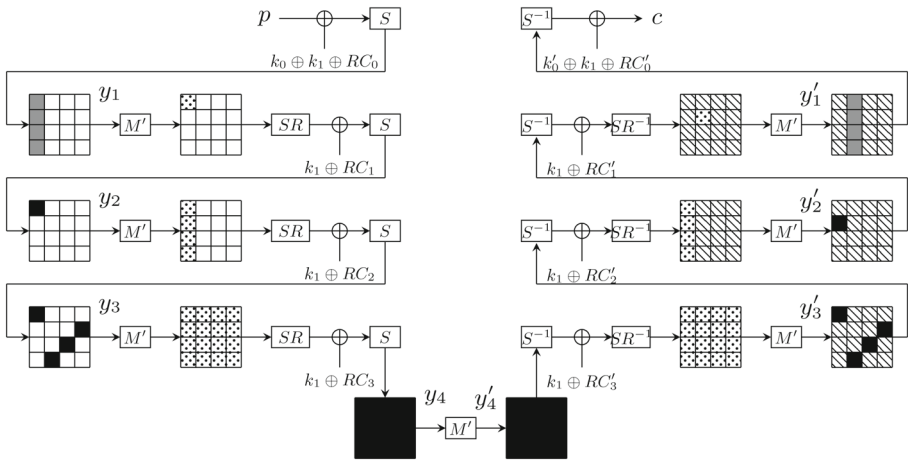


Fig. 19. Step 4. No difference in white nibbles. Nibbles required in online (resp. offline) phase are in gray (resp. black). Differences in dotted nibbles are known during the offline phase. Hatched nibbles play no role.

### Step 7

The missing 48 bits of the key can be exhausted.

### Complexity

The data complexity is  $2^{16}$  chosen plaintexts. The memory complexity is around  $4 \times 2^{48}$  60-bit sequences which is equal to  $2^{52.9}$  bytes. The time complexity is dominated by steps 2/3, which is equivalent to  $2^{64} \times 2^4 \times (42 - 4)/(8 \times 16) \simeq 2^{66.25}$  encryptions.

### References

- [1] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems. *J. CRYPTOLOGY* **4**(1), 3–72(1991).
- [2] M. Matsui, Linear cryptoanalysis method for DES cipher. in *Advances in Cryptology—EUROCRYPT 93, Workshop on the Theory and Application of Cryptographic Techniques, Lofthus, Norway, May 23–27, 1993*, (Proceedings, 1993), pp. 386–397
- [3] H. Demirci, A.A. Selçuk, A meet-in-the-middle attack on 8-round AES. in *Fast Software Encryption*, (Springer, 2008), pp. 116–126
- [4] N. Semiconductors, The PRINCE challenge (2014). [https://www.emsec.rub.de/research/research\\_startseite/prince-challenge/](https://www.emsec.rub.de/research/research_startseite/prince-challenge/)
- [5] J. Borghoff, A. Canteaut, T. Güneysu, E.B. Kavun, M. Knezevic, L.R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, et al, PRINCE—a low-latency block cipher for pervasive computing applications. in *Advances in Cryptology—ASIACRYPT 2012*. (Springer, 2012), pp. 208–225
- [6] A. Biryukov, L. Perrin, State of the art in lightweight cryptography. [http://cryptolux.org/index.php/Lightweight\\_Cryptography](http://cryptolux.org/index.php/Lightweight_Cryptography)
- [7] H. Soleimany, C. Blondeau, X. Yu, W. Wu, K. Nyberg, H. Zhang, L. Zhang, Y. Wang, Reflection cryptanalysis of PRINCE-like ciphers. in S. Moriai, ed., *Fast Software Encryption - 20th International Workshop, FSE 2013*, Singapore, March 11–13, 2013. Revised Selected Papers. Vol. 8424 of Lecture Notes in Computer Science (Springer, 2013), pp. 71–91. [https://doi.org/10.1007/978-3-662-43933-3\\_5](https://doi.org/10.1007/978-3-662-43933-3_5)
- [8] J. Jean, I. Nikolić, T. Peyrin, L. Wang, S. Wu, Security analysis of prince. in *Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11–13, 2013*. Revised Selected Papers. Vol. 8424., (Springer, 2014), **92**
- [9] L. Li, K. Jia, X. Wang, Improved meet-in-the-middle attacks on aes-192 and prince. Cryptology ePrint Archive, Report 2013/573 (2013). <http://eprint.iacr.org/>
- [10] A. Canteaut, T. Fuhr, H. Gilbert, M. Naya-Plasencia, J.R. Reinhard, Multiple differential cryptanalysis of round-reduced PRINCE (full version). Cryptology ePrint Archive, Report 2014/089 (2014). <http://eprint.iacr.org/>
- [11] P. Morawiecki, Practical attacks on the round-reduced PRINCE. *IET Inf. Secur.* **11**(3), 146–151 (2017).
- [12] C. Rechberger, Update on the 10000 euro PRINCE cipher-breaking challenge: Results of round-1 (2014). <http://crypto.2014.rump.cr.yt.to/d037206eda8f9278cef1ea26cd62e51f.pdf>
- [13] O. Dunkelman, N. Keller, A. Shamir, Improved single-key attacks on 8-round AES-192 and AES-256. in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5–9, 2010*. Proceedings. pp. 158–176 (2010)
- [14] P. Derbez, P. Fouque, Exhausting Demirci-Selçuk meet-in-the-middle attacks against reduced-round AES. in *Fast Software Encryption - 20th International Workshop, FSE 2013*, Singapore, March 11–13, 2013. Revised Selected Papers. pp. 541–560 (2013)
- [15] P. Derbez, P.A. Fouque, J. Jean, Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. in T. Johansson, P.Q. Nguyen, eds., *EUROCRYPT*. Vol. 7881 of Lecture Notes in Computer Science., (Springer, 2013), pp. 371–387

- [16] F. Abed, E. List, S. Lucks, On the security of the core of prince against biclique and differential cryptanalysis. Cryptology ePrint Archive, Report 2012/712 (2012). <http://eprint.iacr.org/>
- [17] A. Biryukov, A. Shamir, Structural cryptanalysis of SASAS, in B. Pfitzmann, ed., *Advances in Cryptology—EUROCRYPT 2001*. Vol. 2045 of Lecture Notes in Computer Science. (Springer, Berlin, Heidelberg, 2001), pp. 395–405
- [18] I. Mironov, L. Zhang, Applications of sat solvers to cryptanalysis of hash functions. In Biere, A., Gomes, C., eds.: *Theory and Applications of Satisfiability Testing—SAT 2006*. Volume 4121 of Lecture Notes in Computer Science. (Springer, Berlin, Heidelberg, 2006), pp. 102–115
- [19] N. Eén, N. Sörensson, An extensible SAT-solver. in *Theory and applications of satisfiability testing*, (Springer, 2004), pp. 502–518
- [20] A. Biryukov, Analysis of involuntional ciphers: Khazad and Anubis. in *Fast Software Encryption*, (Springer, 2003), pp. 45–53
- [21] J.H. Moore, G.J. Simmons, Cycle structure of the DES with weak and semi-weak keys. in *Advances in Cryptology—CRYPTO'86*, (Springer, 1987), pp. 9–32
- [22] O. Dunkelman, N. Keller, A. Shamir, Minimalism in cryptography: The Even-Mansour scheme revisited. in *Advances in Cryptology—EUROCRYPT 2012*. (Springer, 2012), pp. 336–354

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.