Cryptanalysis of PRINCE with Minimal Data

Shahram Rasoolzadeh and Håvard Raddum(✉)

Simula Research Laboratory, Fornebu, Norway
haavardr@simula.no

Abstract. We investigate two attacks on the PRINCE block cipher in the most realistic scenario, when the attacker only has a minimal amount of known plaintext available. The first attack is called Accelerated Exhaustive Search, and is able to recover the key for up to the full 12-round PRINCE with a complexity slightly lower than the security claim given by the designers. The second attack is a meet-in-the-middle attack, where we show how to successfully attack 8- and 10-round PRINCE with only two known plaintext/ciphertext pairs. Both attacks take advantage of the fact that the two middle rounds in PRINCE are unkeyed, so guessing the state before the first middle round gives the state after the second round practically for free. These attacks are the fastest until now in the known plaintext scenario for the 8 and 10 reduced-round versions and the full 12-round of PRINCE.

Keywords: PRINCE · Lightweight cipher · Cryptanalysis · Exhaustive search · Meet-in-the-middle

1 Introduction

Designing ciphers that require only a minimum of resources in implementations is known as lightweight cryptography. Several lightweight block and stream ciphers have been proposed, and their design and analysis have been a very active area of research the last decade. PRINCE is a prominent example of a lightweight block cipher, and has received much attention since it was proposed in 2012.

One reason for the interest in cryptanalysis of cryptanalytic results in clearly defined settings. PRINCE' innovative structure has also attracted cryptanalysts to investigate this block cipher. This has resulted in a variety of cryptanalysis in different models, including single key, related key and physical attacks. As the designers did not claim any security in either related key or physical attack models, we focus on the normal single key mode.

Previous works on PRINCE in the single key setting include some attacks on the PRINCE$_{core}$ [3,4,7,8,11] or attacks with change on the original structure [5,8] or an attack in the multi-user case [9]. The attacks which investigate the original structure of PRINCE involves a variety of atttacks, such as integral [4,13], sieve-in-the-middle (SITM) [6], meet-in-the-middle (MITM) [7,12], differential [8,12], and time-memory or time-data-memory trade-off attacks [4,10]. All of these attacks except one exhaustive search-like attack in [4] are chosen

D. Pointcheval et al. (Eds.): AFRICACRYPT 2016, LNCS 9646, pp. 109–126, 2016.
DOI: 10.1007/978-3-319-31517-1_6

Table 1. Summary of cryptanalytic results on PRINCE

Mode	Rounds	Time	Data	Memory	Technique	Ref.
CP	4	2^{64}	2^{4}	2^{4}	Integral	[4]
	4	2^{11}	2^{7}	2^{4}	Integral	[13]
	4	2^{28}	$2^{5.58}$	2^{4}	Integral	[13]
	6	2^{64}	2^{16}	2^{16}	Integral	[4]
	6	2^{41}	$2^{18.58}$	2^{16}	Integral	[13]
	6	$2^{32.9}$	$2^{14.9}$	$\ll 2^{27}$	Differential/Logic	[12]
	6	$2^{33.7}$	2^{16}	$2^{31.9}$	MITM	[12]
	8	2^{60}	2^{53}	2^{30}	MITM	[7]
	8	$2^{50.7}$ *	2^{16}	$2^{84.9}$	MITM	[12]
	8	$2^{65.7}$ *	2^{16}	$2^{68.9}$	MITM	[12]
	10	$2^{60.62}$	$2^{57.94}$	$2^{61.52}$	Multiple Differential	[8]
	10	2^{68} *	2^{57}	2^{41}	MITM	[12]
KP	4	2^{43}	2^{5}	?	? **	[2]
	6	2^{101}	2^{6}	?	? **	[2]
	8	2^{124}	2	2^{20}	SITM	[6]
	8	$2^{122.74}$	2	negl.	Acc. Exh. Search	3.1
	8	$2^{109.34}$	2	$2^{65.03}$	MITM	4.1
	10	$2^{124.06}$	2	negl.	Acc. Exh. Search	3.2
	10	$2^{122.15}$	2	$2^{53.3}$	MITM	4.2
	12	$2^{125.47}$	2	negl.	Exh. Search	[4]
	12	$2^{125.14}$	2	negl	Acc. Exh. Search	3.3

* Online Time

** Attacks reported by Derbez, but not published yet.

plaintext attacks. There is also a known plaintext attack on a reduced-round version in [2] reported by Patrick Derbez but not published yet. A summary of these attacks are given in Table 1.

In this paper we investigate attacks where we assume the attacker only has a minimal amount of known plaintext available, typically only two known plaintext/ciphertext pairs. This is the most realistic scenario, so the results reported here should apply to most implementations. When we have so little data, integral attacks or attacks that rely on statistical biases can not be used, so we are left with algebraic attacks or guess/verify types of attacks. We will focus on the last type of attack, and look at two different attacks of the guess/verify kind that both of them are based on guessing the states right before and after two middle round of the cipher.

The first we call Accelerated Exhaustive Search, and as the name suggests in essence it is an exhaustive search for the key. However, we will show how to exploit certain properties and make several shortcuts when guessing, so the

resulting complexity for this attack becomes significantly lower than in a straightforward exhaustive search. For the full 12-round PRINCE Accelerated Exhaustive Search has a complexity that is lower than the security claim given by the designers (about 1.8 times faster).

The second attack we investigate is a MITM attack. In contrast to [12], but similar to Accelerated Exhaustive Search, our MITM attack by breaking the whole cipher to two smaller sub-ciphers get a 2-dimensional MITM attack [16,17]. We show that the two dimensions can be treated in parallel in PRINCE due to the reflection property, so we can do matching in both sides at the same time and only need to build one big table of values to match instead of two. The main result of this part of the paper is that 10-round PRINCE can be efficiently attacked (with respect to designers' security claim) using only two known plaintexts. Moreover, we get a lower time/data trade-off value $T * D$ than the one reported in [12].

This paper is organized as follows. Section 2 presents a brief description of PRINCE. In Sect. 3 we outline the Accelerated Exhaustive Search attacks and Sect. 4 presents the MITM attacks. Finally, Sect. 5 concludes the paper.

2 PRINCE Block Cipher

PRINCE [1] is a lightweight block cipher with a block size of 64 bits and two keys that both have length 64 bits. It has an FX construction where one of the keys (K_0) are used for whitening and the other one (K_1) is used as a round key for the core of the structure (see Fig. 1). We denote the plaintext/ciphertext pair of PRINCE by P/C, and the corresponding input/output of the PRINCE$_{core}$ function by P'/C'. These variables are related through the following equations.

$$P' = P \oplus K_0, \tag{1}$$

$$C' = C \oplus K_0', \tag{2}$$

where K_0' is the following linear mapping of K_0

$$K_0' = L(K_0) = (K_0 \ggg 1) \oplus (K_0 \gg 63). \tag{3}$$

For any FX constructed block cipher with a linear mapping between the whitening keys (K_0/K_0'), having a known pair of P/C gives us some information about P'/C' of the core structure which is independent from the whitening keys. This is summarized in the following lemma.

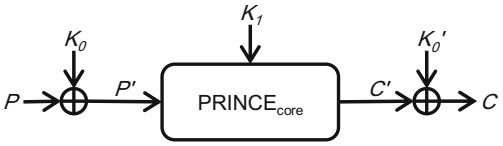

Fig. 1. PRINCE FX construction

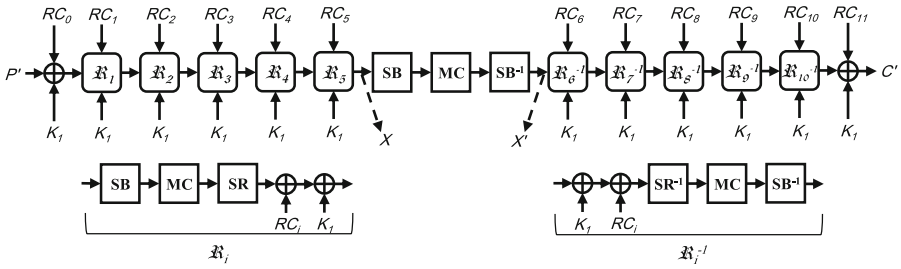

Fig. 2. PRINCE core

Lemma 1. *For a P/C pair and its corresponding P'/C' in the FX construction block cipher which uses a linear mapping L between whitening keys, the following equation holds:*

$$L(P') \oplus C' = L(P) \oplus C \tag{4}$$

Proof. We can eliminate K_0 from (1) and (2), first by applying the $L(.)$ transformation to (1), an then summing up these two equations, which results in (4). So for each pair of P/C we can compute the value of $L(P') \oplus C'$, which is independent of K_0 and K_0'. □

The PRINCE$_{core}$ is a block cipher of its own and similar to AES. It employs an involutive 12 rounds structure which, in the beginning, consists of two *xor*s with the key and a round constant. This is followed by 5 forward rounds, a middle layer, 5 backward rounds and at the end, two more *xor*s with a round constant and the key. Figure 2 shows the schematic view of the PRINCE$_{core}$.

The state can be defined as a 4×4 matrix like for AES, but in PRINCE the cells contain nibbles and not bytes. Each round of the PRINCE$_{core}$ consists of 5 operations: S-box, matrix multiplication, shift row, round constant addition and key addition. These are described as follows.

- **S-box** (SB): Every nibble in the state is replaced using a 4-bit S-box.
- **Matrix Multiplication** (MC): The state is multiplied with an involutive 64×64 binary matrix. More precisely, this large matrix can be expressed as four 16×16 matrices where each of these mixes four nibbles in one column of the state.
- **Shift Row** (SR): It is exactly the same as the shift row operation in the AES. Row i of the state, with row 0 as the top row, is cyclically rotated i positions to the left.
- **Round Constant Addition** (RC): A bit-wise *xor*ing with a round constant RC_i , $i = 0, ..., 11$.
- **Key Addition** (AK): A bit-wise *xor*ing with the key K_1.

The middle rounds contain three layers, SB, MC, SB^{-1} which makes it an involutive keyless transformation. This transformation can also be separated into four smaller transformations, one for each column in the state.

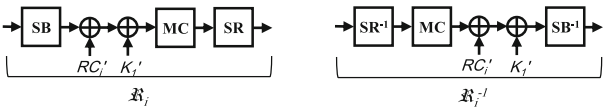

Fig. 3. Modified round function for PRINCE

In the backward rounds, the order of the operations are inverse of the forward rounds, and SB and SR are replaced with SB^{-1} and SR^{-1}. The round constants are also different, but related to the round constants in the forward rounds. The difference $RC_i \oplus RC_{11-i}, i = 0, ..., 11$ is always equal to the constant value $\alpha =$ 0xc0ac29b7c97c50dd.

As a result of this involutive structure of PRINCE$_{core}$, in implementations decryption can use the same circuit as encryption. In decryption mode the key only needs to be *xor*ed with α, i.e.

$$C' = PRINCE_{core}(P', K_1) \Longleftrightarrow P' = PRINCE_{core}(C', K_1 \oplus \alpha). \tag{5}$$

This property is called α-reflection.

To ease the analysis in this paper we define an equivalent key for PRINCE$_{core}$, which is equal to

$$K_1' = SR^{-1}(MC(K_1)). \tag{6}$$

As will be shown in the next sections, using K_1' allows us to simplify the equations used. When we use this equivalent key, we position the AK layer between the SB and MC layers of the round to get an equivalent description of PRINCE$_{core}$ (see Fig. 3). Clearly, by recovering K_1' we can recover K_1.

Finally, as shown in Fig. 2, we denote the internal states exactly before and after the middle rounds by X and X', respectively. Given X, the value of X' can be computed directly, since there is no key involved between X and X'. By using the modified round function with K_1' instead of K_1, we can also expand these keyless rounds by two SR and two MC operations. The X/X' states will be used frequently in the attacks presented in the next sections.

3 Accelerated Exhaustive Key Search

In this section we will describe how to perform an accelerated exhaustive key search on PRINCE. Our way of doing this will be faster than a straight-forward exhaustive key search. By a straight-forward exhaustive key search we mean the attack where we guess a key, fully encrypt a known plaintext, and checks if it matches the given ciphertext. Our attack involves guessing the middle state X and compute the corresponding X'. Knowing a value for K_1' and X in PRINCE$_{core}$ we can easily compute P' and deduce K_0 from (1).

For a plaintext P and the corresponding ciphertext C we will guess the value of X/X' occurring for P and C. For each of the 2^{64} possible X/X'-values, we will search for candidates for K_1'. For each X/X'-value there will be one value of K_1'

in average that will produce P'- and C'-values that will match the given right-hand side in (4). The P'-value computed from this K_1' is then computed and used to deduce K_0. So for each X/X' guess we can expect one (K_0, K_1) candidate. This candidate for the full key can be tried on one other plaintext/ciphertext pair, and if it matches it should be the correct key.

At the outset this looks like an attack with complexity equal to exhaustive key search, but we will show below that the number of S-box look-ups needed to find the (K_0, K_1) candidates can be significantly smaller than in a straight-forward exhaustive key search. Similarly to the biclique attack on AES [14] we count the number of S-box applications we need to use in the attack, and evaluate how many full encryptions this amounts to by trading one round for 16 S-box look-ups. It is argued that a large majority of the time spent in an encryption is used on S-boxes so this trade-off should give rather accurate results.

Our analysis tries to minimise the number of S-box look-ups needed, and the results show that the full PRINCE can be attacked with complexity equal to $2^{125.14}$ encryptions using 2 known plaintexts. This is a little lower than the 2^{127-d}-claim made by the designers when using 2^d texts [1, p. 6].

3.1 Accelerated Exhaustive Search for 8-Round PRINCE

Assuming known X/X', the strategy is to guess on the values of the nibbles in K_1' in such a way that the total number of S-box look-ups for verifying/rejecting a guess becomes minimal. Figure 4 shows the guessing strategy, and which S-boxes that will be computed in the attack. In the following we explain what happens in Fig. 4, focusing on the $P' \leftrightarrow X$ part. Because of the reflective property of PRINCE, the exact same that is done in this part can be done in the $X' \leftrightarrow C'$ part.

Referring to Fig. 4, the nibbles of K_1' will be guessed in alphabetical order, starting with A. When A has been guessed, we have a fixed output of one S-box in round 3. We use one S-box look-up to find the corresponding input, and store this input value. Next, we guess on the value of B, and find the input value to the corresponding third round S-box. To compute these two input values for all the 2^8 possible values (A, B), we must do $2^4(1 + 2^4(1)) = 272$ S-box look-ups since we reuse the stored input for A. This is less than the 512 S-box applications we would have to do in a straight-forward exhaustive search on these two S-boxes. We continue with C and D, storing the input of the third round S-box for each guess.

After D has been guessed we have enough known nibbles between MC and SR in round 2 to go backwards through MC and find the input. At this point we have already guessed the A-value of K_1', so we can add this to the top left nibble and compute the input to the top left S-box in round 2. This is indicated with the state with a single D in this position. The total number of S-box look-ups needed for finding this nibble for all possible values of A, B, C, D is given by the expression

$$2^4(1 + 2^4(1 + 2^4(1 + 2^4(2)))) \tag{7}$$

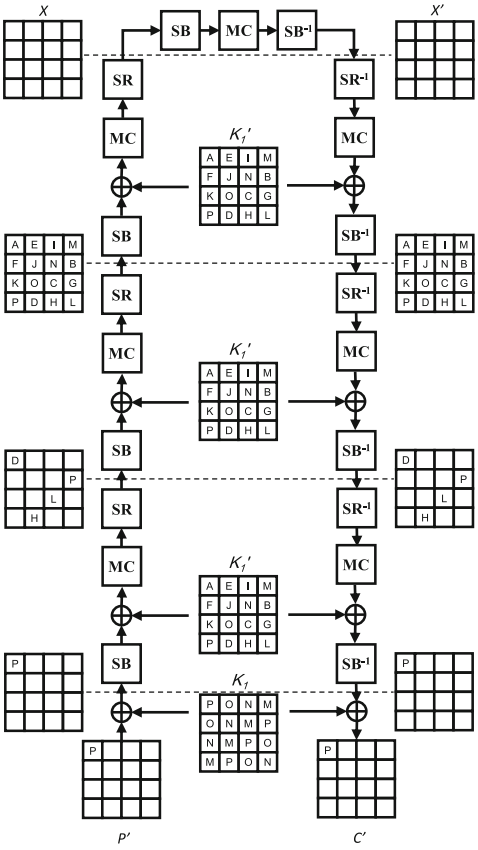

Fig. 4. Order of guessing K_1'-nibbles on 8-round PRINCE.

The 2 in the innermost bracket is because we do S-box look-ups in both round 3 and round 2. We continue in this way, guessing the values of $E - P$ in order, storing the inputs to S-boxes in rounds 2 and 3 whenever they can be computed. The letters in the states indicate which nibbles can be computed after which guess. Note that we do not evaluate all S-box inputs in round 2, only the four indicated with letters.

The number of S-box look-ups needed for computing all indicated nibbles on the $P' \leftrightarrow X$ side when cycling through all possible values of $A - P$ is given by

$$2^4(1 + 2^4(1 + 2^4(1 + 2^4(2 + 2^4(\ldots(1 + 2^4(2))\ldots))))), \tag{8}$$

where every fourth starting number in the brackets is a 2. This number should be multiplied with 2 to count all S-box look-ups for both $P' \leftrightarrow X$ and $X' \leftrightarrow C'$ sides.

Verifying a Guess: When we have made a full guess for K_1' and reached the bottom state in Fig. 4, we are in a position to verify whether the guess was

correct or not. We first apply two S-box look-ups to find the top left nibble of $P' = (p_{63}, \ldots, p_0)$ and $C' = (c_{63}, \ldots, c_0)$. The four nibbles we have learned are $(p_{63}, p_{62}, p_{61}, p_{60})$ and $(c_{63}, c_{62}, c_{61}, c_{60})$. From the definition of L, the first nibble in $L(P') \oplus C'$ is $(p_0 \oplus c_{63}, p_{63} \oplus c_{62}, p_{62} \oplus c_{61}, p_{61} \oplus c_{60})$. Only p_0 is unknown, so we can check the current guess against three bits of the constant $L(P) \oplus C$. If our current guess matches the three bits we evaluate two more S-boxes to learn $(p_{59}, \ldots, p_{56})$ and $(c_{59}, \ldots, c_{56})$. We can now check $(p_{60} \oplus c_{59}, \ldots, p_{57} \oplus c_{56})$ against the constant $L(P) \oplus C$, a total of four new bits.

Continuing in this way, we evaluate the next pair of S-boxes only if the current guess has matched the given part of $L(P) \oplus C$ so far. Note that if the check matches in the first four nibbles, we have to calculate another four nibbles in the bottom state of Fig. 4 before applying the next pair of S-boxes for verification. In these cases we therefore have to apply a total of 10 S-box look-ups instead of 2.

With this analysis we can estimate the number of S-box look-ups needed to verify/reject a guess. This number is given as

$$2\times(1+2^{-3}+2^{-7}+2^{-11}+5\times 2^{-15}+2^{-19}+2^{-23}+2^{-27}+5\times 2^{-31}+\ldots+2^{-63}) \quad (9)$$

Exploiting the α-Reflection Property: The fact that PRINCE is a reflection cipher can be exploited to reduce the amount of guessing. A given value x for the middle state X and a given value k for K_1' will determine particular values p' for P' and c' for C'. Let this be denoted as

$$(x, k) \rightarrow (p', c').$$

Because X and X' are related through an involution, if we chose x' for X, we will get x as a value for X'. PRINCE$_{core}$ is a reflection cipher where decryption is done by encrypting c' with $\alpha \oplus k$. We then know

$$(x', k + \alpha) \rightarrow (c', p'),$$

without having to evaluate all the S-boxes over again. So when we compute the first nibbles of P' and C' and check the first bits of $L(P') \oplus C'$, we can at the same time check $L(C') \oplus P'$. In other words, we check both (x, k) and $(x', k \oplus \alpha)$ at the same time and in this way cut the search space in half.

This can be implemented by enumerating the X-values as $x_i = i$, and do the guessing of the X-values in the natural order $x_0, x_1, \ldots$. We try all keys k for each x_i. When we reach an x_i such that $x_i' < x_i$, we simply skip this x_i because all values k (or rather, $k \oplus \alpha$) have been tried when we had x_i' as a value for X. In this way we only need to try 2^{63} values for the middle state X.

Complexity: When we check nibbles of both $L(P') \oplus C'$ and $L(C') \oplus P'$ for a match against $L(P) \oplus C$, the probability of getting a match which will invoke further S-box look-ups doubles. So the final expression for the number of S-box look-ups needed for verifying a guess becomes

$$2 \times (1 + 2^{-2} + 2^{-6} + 2^{-10} + 5 \times 2^{-14} + 2^{-18} + \ldots + 2^{-62}) = 2 \times 1.2669. \quad (10)$$

This number should be added to the innermost bracket in (8) to give the final expression for the number of S-box look-ups needed to do the accelerated exhaustive search for each X/X'-value:

$$2 \times \left[2^4(1 + 2^4(1 + 2^4(\ldots(1 + 2^4(2 + 1.2669))\ldots)))\right] = 2^{66.74}. \quad (11)$$

We will repeat this for each of 2^{63} values for X, bringing the grand total of S-box look-ups to $2^{129.74}$. Equating 16 S-box look-ups with one round of PRINCE and eight rounds for one encryption, this amounts to a complexity of $2^{122.74}$ encryptions to find the full key (K_0, K_1).

One thing we have glossed over so far in our analysis is the number of S-box look-ups needed to compute X' from X. This is 32 for each of the 2^{63} X-values, so we should add 2^{68} to the total above. The 2^{68} is a negligible addition, so the complexity remains at $2^{122.74}$.

3.2 Accelerated Exhaustive Search for 10-Round PRINCE

The accelerated exhaustive search on 10-round PRINCE is similar to the attack in the previous section, but there are a few differences. One difference is that we have to apply another 16 S-boxes on each of the $P' \leftrightarrow X$ and $X' \leftrightarrow C'$ branches. Another is that we will guess the nibbles of K_1' in a different order than in the 8-round attack. The reason for this is to minimize the value of the expression similar to (8) that applies to the 10-round version. To minimize this value, we want the starting numbers in the brackets to be larger in the outer brackets, and smaller the further into the brackets we get. The order of guessing we have found that minimizes this value is shown in Fig. 5, where we also can see which nibbles that can be computed in the states after each guess.

The expression for counting the number of S-box look-ups will have 16 nested brackets, the first for guessing the A-nibble, the next for B, etc. up to P for the innermost bracket. The starting number in each bracket is the number of new S-box look-ups we can do, and store, after each guess. We must compute the full states at the input to rounds 3 and 4, but only 4 nibbles in the input to round 2, so these numbers will add up to 36. By counting the number of $A, B, C, \ldots, P$ nibbles in the cipher states in Fig. 5, the sequence of starting numbers in the brackets are

$$1, 1, 1, 2, 2, 2, 2, 1, 1, 3, 2, 2, 1, 4, 2, 9.$$

The cost of verifying/rejecting a guess is exactly the same as in the 8-round attack. Multiplying with 2 to cover both the $P' \leftrightarrow X$ and $X' \leftrightarrow C'$ branches, the total number of S-box look-ups for doing the accelerated exhaustive search on 10-round PRINCE for one X-value is

$$2 \times \left[2^4(1 + 2^4(1 + 2^4(1 + \ldots + 2^4(2 + 2^4(9 + 1.2669))\ldots)))\right] = 2^{68.38} \quad (12)$$

Repeating this for all 2^{63} values of X, the total S-box look-ups in the whole attack will be $2^{131.38}$ that is equal to $2^{124.06}$ 10-round PRINCE encryptions.

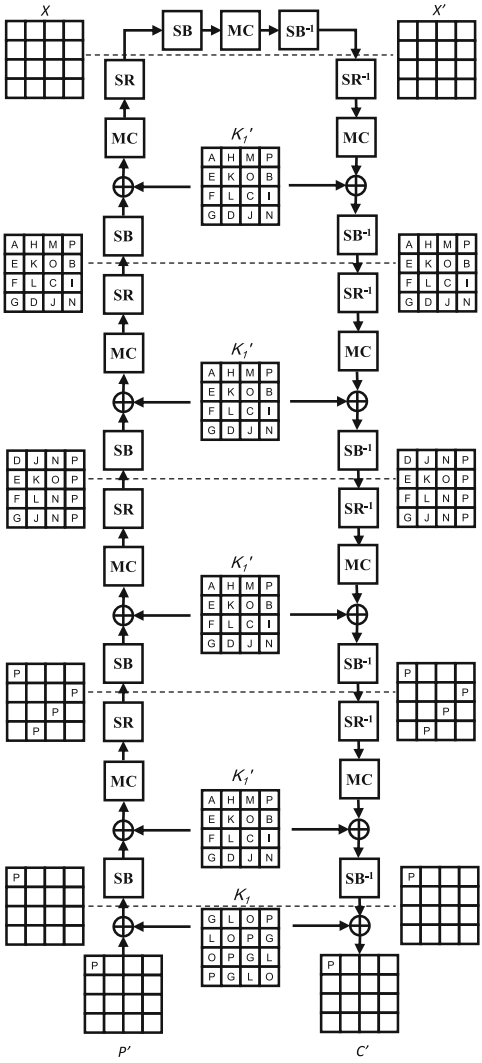

Fig. 5. Order of guessing K_1'-nibbles on 10-round PRINCE.

3.3 Accelerated Exhaustive Search for Full 12-Round PRINCE

For the full PRINCE we can guess the nibbles of K_1' in the same order as for the 10-round version. The expression for counting the total number of S-box look-ups for the $P' \leftrightarrow X$ branch is the same as in (12), except that we must add 16 to the innermost bracket. The total number of S-box look-ups is then

$$2 \times \left[2^4(1 + \ldots + 2^4(25 + 1.2669)\ldots)\right] = 2^{69.72} \tag{13}$$

Repeating for 2^{63} X-values amounts to $2^{132.72}$ S-box look-ups in the total attack, which is equal to $2^{125.14}$ PRINCE encryptions.

The attack uses only 2 known plaintexts, and the security claim given by the designers in this case is that the attacker must use an effort equivalent to 2^{126} PRINCE encryptions to find the secret key. Expecting to find the key half-way through the search, accelerated exhaustive search breaks this bound with an average complexity of $2^{124.14}$ encryptions for finding the secret key.

4 Meet-in-the-Middle Attack on PRINCE

In this section we will briefly introduce the Meet-in-the-Middle (MITM) attack and the technique of guessing only non-linearly involved key nibbles and then we explain the idea of how we do MITM cryptanalysis on PRINCE.

The basic MITM attack is a generic technique presented by Diffie and Hellman to cryptanalyse DES [15]. Despite the fact that this technique is arguably much less common than differential or linear attacks on block ciphers, there are some extensions and applications of this attack to specific primitives which are more successful than differential and linear attacks.

Let $E_{i,j}(S, K_f)$ denote the partial encryption of the state S, beginning from the start of round i and ending at the start of round j, where K_f is a particular sequence of subkeys corresponding to these $j-i$ rounds. Similarly, let $D_{j,i}(S, K_b)$ denote the partial decryption of S, beginning from the start of round j and ending at the start of round i, where K_b is the sequence of subkeys corresponding to these $j - i$ rounds. The main idea of a MITM attack is that the subkeys in both parts of the cipher can be guessed separately. First, the attacker guesses K_f and computes $E_{0,r}(P, K_f)$ for a known plaintext P. Next, he guesses K_b and computes $D_{R,r}(C, K_b)$ for the corresponding ciphertext. If

$$E_{0,r}(P, K_f) = D_{R,r}(C, K_b), \tag{14}$$

then the guessed values for K_f and K_b are candidates for representing the correct secret key.

Linearly Involved Key Bits: In the technique of guessing only non-linearly involved key bits, we do not guess the key bits which are only *xor*ed to the bits we use for matching. For example, in the MITM attack we can write:

$$\begin{cases} E_{0,r}(P, K_f) = E'_{0,r}(P, K'_f) \oplus L_f K''_f, \\ D_{R,r}(C, K_b) = D'_{R,r}(C, K'_b) \oplus L_b K''_b. \end{cases} \tag{15}$$

Here K'_f and K'_b are subsets of K_f and K_b such that $E_{0,r}(P, K_f)$ and $D_{R,r}(C, K_b)$ are non-linearly dependent on them. L_f and L_b are binary matrices, only *xor*ing some bits of K''_f or K''_b to the state bits.

When the key schedule is linear and K'_f and K'_b together determine the user-selected key, we can always find two binary matrices L'_f and L'_b of full rank which satisfy

$$L'_f \cdot K'_f \oplus L'_b \cdot K'_b = L_f \cdot K''_f \oplus L_b \cdot K''_b. \tag{16}$$

That is, K''_f and K''_b can be expressed as linear combinations of K'_f and K'_b bits. Instead of checking equation (14) for the whole K_f and K_b, we can then check for

$$E'_{0,r}(P, K'_f) \oplus L'_f \cdot K'_f = D'_{R,r}(C, K'_b) \oplus L'_b \cdot K'_b, \tag{17}$$

where the left and right hand sides can be calculated by K'_f and K'_b in the forward and backward sides of a MITM attack, respectively. This technique enables us to reduce the number of guessed key bits in each side of a MITM attack.

MITM on PRINCE: For PRINCE, because of the reflection property, knowing the value of X, we can break the whole $R = 2r + 2$-round structure into two equal sub-ciphers with r rounds and a linearly related key. Assume we know the values of X/X' for a P/C pair. Then we have two equations for the same sub-ciphers:

$$\begin{cases} F(P \oplus K_0, K_1) = X, \\ F(C \oplus K'_0, K_1 \oplus \alpha) = X', \end{cases} \tag{18}$$

where $F(S, K)$ denotes the encryption function of state S under key of K, for r forward rounds of the PRINCE$_{core}$ structure.

As finding a MITM matching for r rounds is easier than for $2r+2$ rounds, our idea is that for a known P/C pair we guess a value of X/X' and break the cipher into two smaller sub-ciphers and do a MITM attack on each of the sub-ciphers in parallel. From the P/C sides, we will guess some bits of K_0 (denoted by K_w) and some nibbles of K_1 (denoted by K_s). For a guessed value of K_w and K_s we will calculate m bits from a state in the middle of each r-round sub-ciphers ($2m$ bits in total) and save them in a table. From the middle of the whole structure we will guess values of X/X' and some nibbles of K'_1 (denoted by K_m) that allows us to calculate the same $2m$ bits in the middle of each r-round sub-cipher. Then we can check equality of m bits in each sub-cipher as defined by (18).

As both K_s and K_m are derived from K_1, they may have some common information bits, which we denote as K_c. Guessing values of K_c before any other values will help us to reduce the complexity of the attack. The algorithm of the attack is described in Algorithm 1, where $E_{0,i}(.,.)$ and $D_{r,i}(.,.)$ denotes partially encryption or decryption functions for the r-round sub-cipher defined by $F(.,.)$ in (18).

This extension of the MITM attack may be considered as a multidimensional (MD) MITM attack [16–18], because we break the whole cipher into two sub-ciphers by guessing a full state in the middle. On the other hand, here we do two MITM matchings in parallel with each other, while in a MD MITM attack matchings happens serially, one after another.

The data complexity of the attack is 2 known plaintexts. The memory complexity of the attack is storing the table $\mathcal{T}$ which will cost $2^{|K_w|+|K_s-K_c|}$ words of memory.

Algorithm 1. MITM attack on PRINCE

for $k_c \in K_c$ **do**
  for $k_w \in K_w$ **do**
    for $k_s \in (K_s - K_c)$ **do**
      Compute $v_1 = E_{0,i}(P, (k_w, k_c, k_s))$ and $v_2 = E_{0,i}(C, (k_w, k_c, k_s))$;
      Store (k_w, k_s) into a table $\mathcal{T}$ indexed by (v_1, v_2);
    end for
  end for
  for $X \in \mathbb{F}_2^{64}$ **do**
    Compute X';
    for $k_m \in (K_m - K_c)$ **do**
      Compute $v_1' = D_{r,i}(X, (k_c, k_m))$ and $v_2' = D_{r,i}(X', (k_c, k_m))$;
      Find $(k_w, k_s) = \mathcal{T}[v_1', v_2']$ (if it exists);
      Verify/reject the candidate (k_w, k_s, k_m, k_c) against other state bits;
      if (k_w, k_s, k_m, k_c) fits all other state bits **then**
        Check it on another known plaintext/ciphertext pair;
        if (k_w, k_s, k_m, k_c) fits the second plaintext/ciphertext pair **then**
          Return (k_w, k_s, k_m, k_c) as correct key;
        end if
      end if
    end for
  end for
end for

4.1 MITM Attack to 8-Round PRINCE

Guessing the value of X/X' will break 8-round PRINCE into two 3-round subciphers. From the P/C sides, we guess 48 bits in the three leftmost columns of K_0 and K_0' (equal to 49 bits of K_0) and the three leftmost columns of K_1. From the X/X' sides we guess all nibbles of K_1' except nibbles on the secondary diagonal of the state. These 48 bits of K_s and 48 bits of K_m have 32 common information bits denoted by K_c. After guessing the value of the 32 K_c bits, only 16 bits from each set of keys remain for guessing.

Figure 6 shows the procedure of the attack. From the P/C sides we will be able to calculate 9 nibbles of the states before the MC layer in the second and seventh rounds (Gray/White squares are related to computed/un-computed nibbles). From the X/X' sides we can calculate 12 nibbles of the state before the MC layer in the second and seventh rounds, so we can do a matching on 2×9 common nibbles of these states.

In the second and seventh rounds the AK layer is not shown in Fig. 6. As K_s and K_m determine all 64 bits of K_1, we can include it using the technique of equation (17).

Attack Procedure: The attack follows Algorithm 1. In the first stage of the attack we create a table $\mathcal{T}$ from the P/C sides. In the table $\mathcal{T}$ we should save the value of $49 + 16 = 65$ bits (k_w, k_s) indexed by the 72 bits of (v_1, v_2) from the two states before the MC layer. Only a fraction of $2^{65-72} = 2^{-7}$ indexes in $\mathcal{T}$

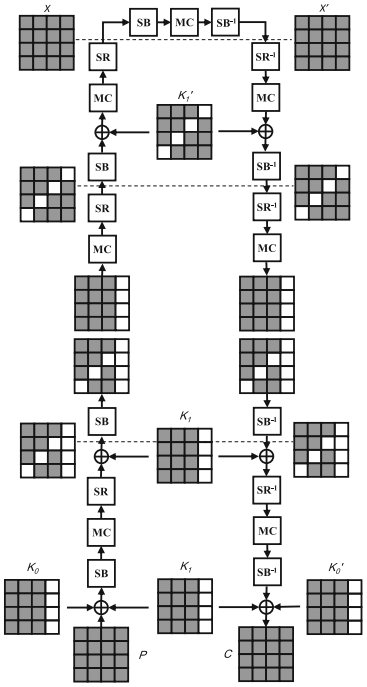

Fig. 6. MITM cryptanalysis of 8-round PRINCE

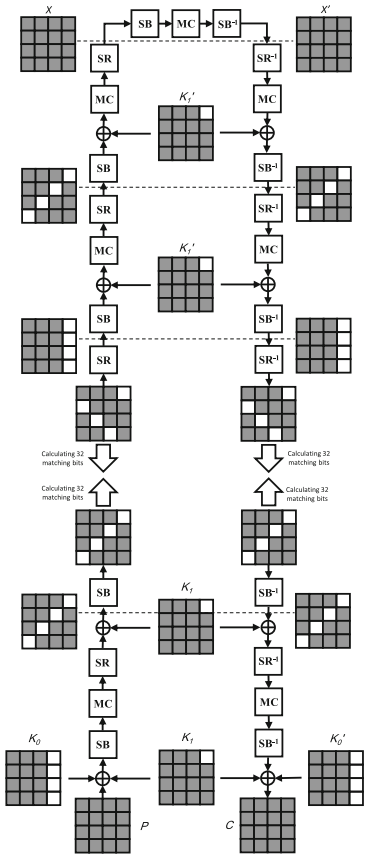

Fig. 7. MITM cryptanalysis of 10-round PRINCE

will then be filled so the storage in the table would be larger than it needs to be. Instead we save the 65 bits of (k_w, k_s) and the 7 last bits of (v_1, v_2) in the cell indexed by the 65 first bits of (v_1, v_2). Then we expect each cell in $\mathcal{T}$ to contain one value.

In the second stage of the attack we evaluate 64 bits of (v_1', v_2') for the guessed values of (X, k_m, k_c) and pick up the content in $\mathcal{T}$ for the index of the 65 first bits of (v_1', v_2'). First we check whether the 7 last bits of (v_1', v_2') are equal to the 7 last bits of (v_1, v_2) or not. If they match we have a candidate for $(k_w, k_s, k_m, k_c) = (k_w, K_1)$, 113 bits of the key, and the middle values X/X'.

Using the values of X/X' and K_1 we calculate the nibbles which we did not evaluate (white squares in Fig. 6), coming from X/X' to plaintext and ciphertext sides. By evaluating them we can do a matching for equality of 3 nibbles at the output of the first round and 3 nibbles of the input to the eighth round. If these 24 bits match, we will evaluate P' and C' which will allow us to find a unique

value for the 15 un-guessed bits of K_0 and also another $32 - 15 = 17$ matching bits. If this matching happens we will have only one candidate for (K_0, K_1) that with probability of 2^{-64} is the correct key. Using another pair of plaintext and ciphertext will verify whether it is the correct key or not.

Complexity: The time complexity of this attack will be dominated by the computation time of the second stage. In the second stage, for each guess of 32 bits k_c, 16 bits k_m and 64 bits X, we must calculate X' from X (32 S-box calls) and 12 nibbles in the middle of second and seventh rounds (12 calls for each). Trading 16 S-box look-ups with one round of PRINCE, this is equal to

$$2^{32+16+64} \times (32 + 2 \times 12) \times \frac{1}{16} \times \frac{1}{8} = 7 \times 2^{108} \tag{19}$$

encryptions of 8-round PRINCE.

As X is the last parameter to guess in the second stage, instead of guessing the whole 64 bits we can guess it one nibble at the time (the same technique used for accelerated key search in Sect. 3) to reduce the time complexity by a fraction of about 0.362. So the final complexities of the attack will be $2^{109.34}$ encryptions for time and 2^{65} 72-bit blocks for memory.

4.2 MITM Attack to 10-Round PRINCE

Guessing the value of X/X' will break 10-round PRINCE into two 4-round subciphers. We guess the same 49 bits of K_0 and K_0' as for the 8-round attack, and all nibbles of K_1 except the one on top of the rightmost column. From the X/X' sides we guess all nibbles of K_1' except the one on top of the rightmost column. These 60 bits of K_s and 60 bits of K_m have 56 common information bits denoted by K_c. After guessing the value of the 56 K_c bits, only 4 bits from each set of keys remain to be guessed.

As Fig. 7 shows, from the P/C sides we are able to calculate 12 nibbles of the states before the MC operation in the second and ninth rounds. From the X/X' sides, we can also calculate 12 nibbles of the states on the other side of the MC layers in the second and ninth rounds. Exactly one nibble in each column is unknown in each state.

Matching Through MC: We have two partially known states on both sides of the MC operation, and we can match these in a similar way to what is done in [19]. Let one column of bits in the input to MC be (x, a, b, c) with output (y, d, e, f), where only the x and y bits are unknown. As will become clear, the exact positions of the unknown bits do not matter. The MC operation on this column gives 4 linear equations in the input and output:

$$\begin{aligned} l_0(x, a, b, c) + y &= 0 \\ l_1(x, a, b, c) + d &= 0 \\ l_2(x, a, b, c) + e &= 0 \\ l_3(x, a, b, c) + f &= 0 \end{aligned} \tag{20}$$

We can eliminate the unknown x and y variables and get two linear equations in the known $a, \ldots, f$:

$$\begin{aligned} l'_0(a,b,c) + l''_0(d,e,f) &= 0 \\ l'_1(a,b,c) + l''_1(d,e,f) &= 0 \end{aligned} \tag{21}$$

Also we will have unique equations for x and y:

$$\begin{aligned} x &= l'_2(a,b,c) + l''_2(d,e,f) \\ y &= l'_3(a,b,c) + l''_3(d,e,f) \end{aligned} \tag{22}$$

Coming from the plaintext side we define the two bits of v_1 relating to this column to be (l'_0, l'_1), and from the X side we define the corresponding two bits of v'_1 to be (l''_0, l''_1). Besides these two values, from the plaintext side we will save l'_3 to compute the unknown value of y later, after matching. From the X side we can evaluate the value of y by computing and adding l''_3 to l'_3. Repeating for the other 15 columns gives a total of 32 bits in v_1 and v'_1 that can be used for matching, plus four l'_3-values.

The same procedure is done in the ninth round, so all together we get 64-bit values for (v_1, v_2) and (v'_1, v'_2) that can be used for matching as described in Algorithm 1. In addition we get 32 stored bits for evaluating the unknown values in the state after the MC layer in both of the second and ninth rounds.

Attack Procedure: The attack follows the same procedure as before, but with different numbers of guessed bits. In the table $\mathcal{T}$ we will save 53 bits of (k_w, k_s) and the 11 last bits of (v_1, v_2), indexed by the 53 first bits of (v_1, v_2). In the second stage, we compute (v'_1, v'_2) from the X/X' sides and do the matching with the corresponding value in $\mathcal{T}$. We then have a candidate value for (k_w, K_1) of 113 bits and known X/X' values. Using the saved l'_3-values we can then compute the rest of K_0 and the un-evaluated nibbles of the states and verify the current guess for correctness.

There will be one (K_0, K_1) candidate surviving for each X/X' guess, which has to be verified against another plaintext/ciphertext pair.

Complexity: The number of bits to guess in the first stage is 15 bits less than the number of bits to guess in the second stage, so the time complexity of this attack will be dominated by the computation time of the second stage. In the second stage, for each guess of 56 bits k_c, 4 bits k_m and 64 bit X, we must calculate X' from X (32 S-box calls) and 12 nibbles in the middle of second and ninth rounds (24 calls for each). Trading 16 S-box look-ups with one round of PRINCE, this is equal to

$$2^{56+4+64} \times (32 + 2 \times 24) \times \frac{1}{16} \times \frac{1}{10} = 2^{123} \tag{23}$$

encryptions of 10-round PRINCE.

Here again, instead of guessing the whole 64 bits of X we can guess it nibble-wise to reduce time complexity by a fraction of 0.553. The final complexities will then be $2^{122.15}$ for time, and 2^{53} 96-bit blocks for memory.

5 Conclusions

In this paper we have shown that PRINCE can be efficiently attacked with respect to the security bound 2^{127-d}, even when having only a minimal amount of known plaintext available. To our knowledge, accelerated exhaustive search is the first reported attack on the full PRINCE with complexity lower than the claim given by the designers.

The practice of counting the number of S-box look-ups needed in an attack and translating this into number-of-encryptions complexity has already been established. It is clear that evaluating S-boxes takes the majority of the time in implementations, but it would be good to get more accurate numbers on exactly how large percentage of the time is spent on S-box look-ups and how much is spent on the linear layers. We have been in contact with some of the designers of PRINCE, and know they are working on producing these numbers. This will give a better scientific foundation for estimating the time complexity, and should allow to report very accurate numbers for this with confidence.

We have also implemented a new meet-in-the-middle attack to PRINCE where we can successfully attack 8 and 10 rounds, again with only two known plaintext/ciphertext pairs. Although meet-in-the-middle attacks have a big memory complexity, it shows that only having a minimum of known plaintext available, 8- and 10-round PRINCE can still be attacked efficiently using a meet-in-the-middle approach. As far as we know, these attacks for the 8 and 10 reduced-round versions are the fastest until now in the known plaintext scenario.

References

1. Borghoff, J., et al.: PRINCE – A low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
2. The PRINCE Teamd: PRINCE Challenge. https://www.emsec.rub.dde/research/research_startseite/prince-challenge/
3. Abed, F., List, E., Lucks, S.: On the security of the core of PRINCE against biclique and differential cryptanalysis. IACR Cryptology ePrint Archive, Report /712, 2012 (2012)
4. Jean, J., Nikolić, I., Peyrin, T., Wang, L., Wu, S.: Security analysis of PRINCE. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 92–111. Springer, Heidelberg (2014)
5. Soleimany, H., Blondeau, C., Yu, X., Wu, W., Nyberg, K., Zhang, H., Zhang, L., Wang, Y.: Reflection cryptanalysis of PRINCE-like ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 71–91. Springer, Heidelberg (2014)
6. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013)
7. Li, L., Jia, K., Wang, X.: Improved meet-in-the-middle attacks on AES-192 and PRINCE, IACR Cryptology ePrint Archive, Report /573, 2013 (2013)
8. Canteaut, A., Fuhr, T., Gilbert, H., Naya-Plasencia, M., Reinhard, J.-R.: Multiple differential cryptanalysis of round-reduced PRINCE. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 591–610. Springer, Heidelberg (2015)

9. Fouque, P.-A., Joux, A., Mavromati, C.: Multi-user collisions: Applications to discrete logarithm, even-mansour and PRINCE. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 420–438. Springer, Heidelberg (2014)
10. Dinur, I.: Cryptanalytic time-memory-data tradeoffs for FX-constructions with applications to PRINCE and PRIDE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 231–253. Springer, Heidelberg (2015)
11. Zhao, G., Sun, B., Li, C., Su, J.: Truncated differential cryptanalysis of PRINCE. Secur. Commun. Netw. **8**, 2875–2887 (2015). Wiley
12. Derbez, P., Perrin, L.: Meet-in-the-middle attacks and structural analysis of round-reduced PRINCE. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 190–216. Springer, Heidelberg (2015)
13. Morawiecki, P.: Practical attacks on the round-reduced PRINCE. IACR Cryptology ePrint Archive, Report /245, 2015 (2015)
14. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
15. Diffie, W., Hellman, M.: Exhaustive cryptanalysis of the NBS data encryption standard. IEEE Comput. Soc. Press **10**(6), 74–84 (1977)
16. Zhu, B., Gong, G.: Multidimensional meet-in-the-middle attack and its applications to KATAN32/48/64. Cryptography and Communications **6**, 313–333 (2014). Springer
17. Boztaş, Ö., Karakoç, F., Çoban, M.: Multidimensional meet-in-the-middle attacks on reduced-round TWINE-128. In: Avoine, G., Kara, O. (eds.) LightSec 2013. LNCS, vol. 8162, pp. 55–67. Springer, Heidelberg (2013)
18. Rasoolzadeh, S., Raddum, H.: Multidimensional meet in the middle cryptanalysis of KATAN. IACR Cryptology ePrint Archive, Report /077, 2016 (2016)
19. Sasaki, Y.: Meet-in-the-middle preimage attacks on AES hashing modes and an application to whirlpool. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 378–396. Springer, Heidelberg (2011)