HAL
archives-ouvertes.fr

Security Analysis of PRINCE

Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, Shuang Wu

To cite this version:

Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, Shuang Wu. Security Analysis of PRINCE. FSE 2013, Mar 2013, Singapore, Singapore. 2013. <hal-00870448>

HAL Id: hal-00870448

https://hal.inria.fr/hal-00870448

Submitted on 7 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Security Analysis of PRINCE

Jérémy Jean[1]*, Ivica Nikolić[2], Thomas Peyrin[2], Lei Wang[2] and Shuang Wu[2]

[1] École Normale Supérieure, France
[2] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
Jeremy.Jean@ens.fr {inikolic,thomas.peyrin,wang.lei,wushuang}@ntu.edu.sg

Abstract. In this article, we provide the first third-party security analysis of the PRINCE lightweight block cipher, and the underlying $\texttt{PRINCE}_{core}$. First, while no claim was made by the authors regarding related-key attacks, we show that one can attack the full cipher with only a single pair of related keys, and then reuse the same idea to derive an attack in the single-key model for the full $\texttt{PRINCE}_{core}$ for several instances of the α parameter (yet not the one randomly chosen by the designers). We also show how to exploit the structural linear relations that exist for PRINCE in order to obtain a key recovery attack that slightly breaks the security claims for the full cipher. We analyze the application of integral attacks to get the best known key-recovery attack on a reduced version of the PRINCE cipher. Finally, we provide time-memory-*data* tradeoffs, that require only known plaintext-ciphertext data, and that can be applied to full PRINCE.

Key words: PRINCE, block cipher, cryptanalysis, related-key boomerang, time-memory-data tradeoff.

1 Introduction

Lightweight cryptography is a new, rapidly developing area of symmetric cryptography that has emerged from the needs of constrained devices. The increasing deployment of such devices in the everyday life has captured the attention of the cryptographic community. It became clear that most of the available cryptographic primitives, both ciphers and hash functions, fail to meet the basic requirements of constrained devices – low cost hardware implementation, as well as low power usage and latency. Thus, so-called lightweight primitives, designed only for these type of devices, have been proposed (and some already have been implemented) in the past several years.

PRINCE [4] is a lightweight cipher published at Asiacrypt 2012, and optimized with respect to latency when implemented in hardware. It is based on Even-Mansour-like construction (so-called FX construction [2, 10]) and it has the interesting feature that one can perform decryption by reusing the encryption process with a slightly different key. This feature, so-called α-reflection property, clearly provides an advantage in implementations requiring both encryption and decryption, but at the same time induces some structure. This structure forced the designers to reduce the security expectations compared to an ideal cipher and they claimed that the security of the cipher is ensured up to 2^{127-n} operations when 2^n encryption/decryption queries are made. This bound is only valid for the single-key model, and the authors made no claim concerning the related-key model (a trivial related-key distinguisher can be built).

Our contributions. In this article, we provide the first third-party analysis of the PRINCE cipher. First, we analyze in Section 3 the resistance of PRINCE in regard to related-key attacks. We emphasize that the designers clearly did not make any claim regarding this attack model. However, while no claim was made in this scenario, the best attack so far was a trivial related-key distinguisher and it was not clear up to what extend an attack can be mounted. We show that with a single pair of related keys, one can recover the whole secret key faster than exhaustive search or faster than the claimed single-key security bound.

Moreover, our related-key attacks are actually interesting not only for the related-key model, but also for the single-key one since we leverage these techniques to show in Section 4 that several choices of values for α lead to an insecure version of $\texttt{PRINCE}_{core}$ in the single-key model. It is to be noted that the designers required $\alpha \neq 0$ to enforce their security claims and the value of α was eventually derived from the fraction part of π. We show that the choice of α is actually sensitive for the security of the cipher.

In Section 5, we exploit the related-key relations verified with probability 1 that exist for PRINCE in order to mount a key recovery attack, slightly breaking the designers claims in the single-key scenario. Namely, we show that one can generically gain a factor $2^{0.6}$ compared to their claims, by only taking into account that the cipher is using the FX construction and has the α-reflection property. While the gain is quite small, it indicates that perhaps a more precise security proof (taking in account the α-reflection property) might be an interesting research problem.

* This collaborative work was done while the first author was visiting CCRG lab of Nanyang Technological University in Singapore.

We explore the application of integral attacks in Section 6 and improve the best known result on a reduced version of PRINCE, providing a 6 rounds key recovery attack with low complexity.

Finally, in Section 7 we propose tradeoffs for PRINCE. We show that due to the specific structure of the cipher, tradeoffs involving data and requiring only known plaintexts-ciphertext are achievable for PRINCE. We start with a Memory-Data tradeoff based on the meet-in-the-middle technique, and improve our results to Time-Memory-Data tradeoff based on the original Hellman's approach.

Our results are summarized in Table 1.

Table 1. Summary of the results on PRINCE and $\texttt{PRINCE}_{core}$.

Cipher	Rounds	Data	Time	Memory	Technique	Reference
PRINCE	4	2^4	2^{64}	2^4	Integral	Section 6
	5	$5 \cdot 2^4$	2^{64}	2^8	Integral	Section 6
	6	2^{16}	2^{64}	2^{16}	Integral	Section 6
	12	2^1	$2^{125.47}$	negl.	Single-Key	Section 5
	12 †	2^{33}	2^{64}	2^{33}	Related-Key	Section 3.1
	12	$MD = N, T = N^{\frac{1}{2}}$			Memory-Data TO	Section 7
	12	$T(MD)^2 = N^2 N^{\frac{1}{2}}$			Time-Memory-Data TO	Section 7
	12	$TMD = NN^{\frac{1}{2}}$			Time-Memory-Data TO	Section 7
$\texttt{PRINCE}_{core}$	4	2^4	2^8	2^4	Integral	Section 6
	5	$5 \cdot 2^4$	2^{21}	2^8	Integral	Section 6
	6	2^{16}	2^{30}	2^{16}	Integral	Section 6
	12 †	2^{39}	2^{39}	2^{39}	RK Boomerang	Section 3.2
	12	2^{41}	2^{41}	negl.	SK Boomerang for Chosen α	Section 4

†: No security claim for related-key attacks
RK: Related-key
SK: Single-key
TO: (Cryptanalytic) Tradeoff

2 Description of PRINCE

PRINCE [4] is a 64-bit block cipher that uses a 128-bit secret key k. The key expansion first divides k into two parts of 64 bits each $k = (k_0||k_1)$, where $||$ denotes the concatenation, and then extends the key material into 192 bits:

$$k = (k_0 \,||\, k_1) \rightarrow (k_0 \,||\, k_0' \,||\, k_1) = (k_0 \,||\, L(k_0) \,||\, k_1), \qquad \text{with: } L(x) = (x \ggg 1) \oplus (x \gg 63). \tag{1}$$

The 64-bit subkeys k_0 and k_0' are used as input and output whitening keys respectively, while k_1 is used as internal key for the core block cipher $\texttt{PRINCE}_{core}$ (see Fig. 1).

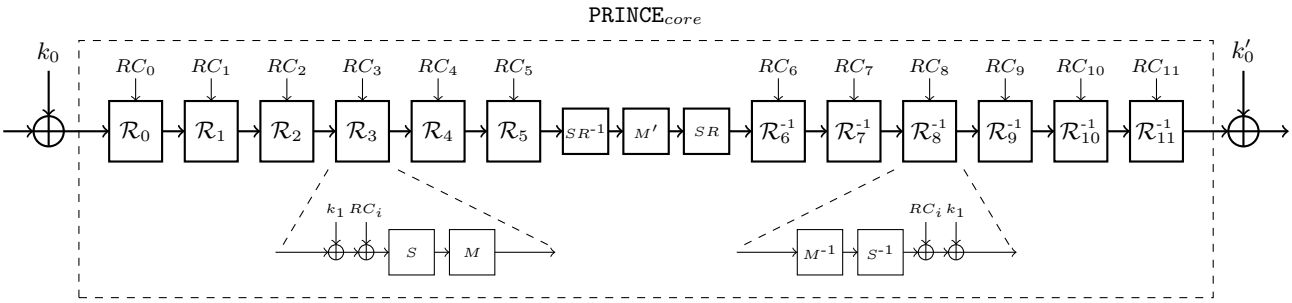

Fig. 1. A schematic view of the PRINCE cipher.

The internal block cipher $\texttt{PRINCE}_{core}$ is a Substitution-Permutation Network composed of 12 rounds. A round function $\mathcal{R}_i$ is defined by the bitwise addition of the 64-bit subkey k_1 and a 64-bit constant RC_i, the

application of a 4-bit Sbox S to each of the 16 4-bit nibbles of the internal state, and finally the multiplication by a linear diffusion matrix M. The encryption of $\texttt{PRINCE}_{core}$ is then composed of the application of the 6 rounds $\mathcal{R}_0, \ldots, \mathcal{R}_5$, the multiplication by a linear diffusion matrix M_{mid}, and finally the application the 6 inverse rounds $\mathcal{R}_6^{-1}, \ldots, \mathcal{R}_{11}^{-1}$:

$$\texttt{PRINCE}_{core} = \mathcal{R}_{11}^{-1} \circ \mathcal{R}_{10}^{-1} \circ \mathcal{R}_9^{-1} \circ \mathcal{R}_8^{-1} \circ \mathcal{R}_7^{-1} \circ \mathcal{R}_6^{-1} \circ M_{mid} \circ \mathcal{R}_5 \circ \mathcal{R}_4 \circ \mathcal{R}_3 \circ \mathcal{R}_2 \circ \mathcal{R}_1 \circ \mathcal{R}_0.$$

The 4-bit S-box S has a maximal differential probability of $p_{max} = 2^{-2}$, and is given by (in hexadecimal display) $S[x] = [\texttt{B,F,3,2,A,C,9,1,6,7,8,0,E,5,D,4}]$. The linear diffusion matrix M is composed of a linear matrix M' and a nibble shifting part SR (similar to a ShiftRows in AES [6]): $M = SR \circ M'$. Then, the linear middle matrix M_{mid} is defined by $M_{mid} = M \circ M' \circ M^{-1} = SR \circ M' \circ SR^{-1}$. We refer to [4] for the complete description of M', but one must remark that its diffusion property ensures that at least 16 Sboxes are active for 4 consecutive round functions.

It is to be noted that $RC_i \oplus RC_{11-i} = \alpha = \texttt{0xc0ac29b7c97c50dd}$ for all $0 \leq i \leq 11$, and since the matrix M' is an involution, this allows to perform the decryption D of PRINCE by simply encrypting with the key $k_1 \oplus \alpha$ instead of k_1 and flipping the whitening keys k_0 with k_0': $D_{(k_0 \,||\, k_0' \,||\, k_1)}(\cdot) = E_{(k_0' \,||\, k_0 \,||\, k_1 \oplus \alpha)}(\cdot)$.

In this article, we see the internal state s of PRINCE as a 4×4 matrix form, where each cell is a nibble, and if we denote $s[i]$ the i-th nibble, $0 \leq i < 16$ from MSB to LSB, it would be located at row $i \pmod 4$ and column $\lfloor i/4 \rfloor$.

3 Related-key attacks

In this section, we describe a related-key attack on the full PRINCE, and a related-key attack on the core block cipher $\texttt{PRINCE}_{core}$. The first one (Section 3.1) uses a single related-key, and the α-reflection property of the core cipher to recover the 128-bit master key with 2^{33} data, 2^{63} operations and 2^{32} memory. The second attack (Section 3.2) uses a related-key differential characteristic with high-probability to mount a boomerang distinguisher on the core block cipher, that can be turned into a key-recovery attack for the 64-bit key k_1 of $\texttt{PRINCE}_{core}$. We have verified experimentally our results – an example of boomerang quartet for the full 12-round $\texttt{PRINCE}_{core}$ is given in Appendix A.

3.1 Related-key attack on full PRINCE with the α-reflection property

We denote in the sequel the secret master key by $k = (k_0, k_1)$ that we aim to recover. We introduce one related-key $k' = (k_0, k_1 \oplus \alpha)$, where α refers to constant defined in Section 2. The attack procedure uses the following distinguisher on the whole core of PRINCE.

Property 1. Let (P, C) be a plaintext/ciphertext pair encrypted under the secret key k by PRINCE, and (P', C') an other plaintext/ciphertext pair from PRINCE with the related key k'. If $C \oplus P' = k_0 \oplus L(k_0)$, then $P \oplus C' = k_0 \oplus L(k_0)$ with probability 1.

Proof. As described in Section 2, PRINCE transforms a plaintext P into the ciphertext $C = E_{k_1}(k_0 \oplus P) \oplus L(k_0)$, where E_{k_1} instantiates $\texttt{PRINCE}_{core}$ with key k_1. For a second plaintext P', we set $C' = E_{k_1 \oplus \alpha}(k_0 \oplus P') \oplus L(k_0)$

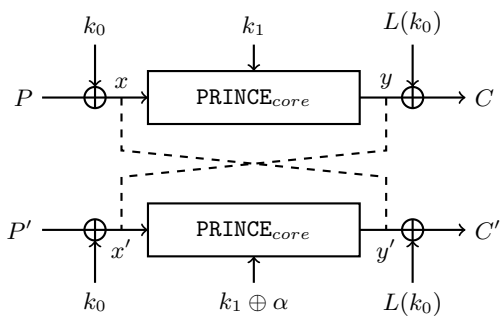

Fig. 2. Related-key distinguisher on full PRINCE.

using the related-key. The condition $C \oplus P' = k_0 \oplus L(k_0)$ actually states that the output of $\texttt{PRINCE}_{core}$ in the first message equals the input of $\texttt{PRINCE}_{core}$ in the second one. Namely, $C \oplus P' = k_0 \oplus L(k_0)$ means $x' = y$ from the notations of Figure 2. Since $y = E_{k_1}(x)$ and $y' = E_{k_1 \oplus \alpha}(x')$, we have $x = y'$, which gives $P \oplus C' = k_0 \oplus L(k_0)$. □

From this distinguisher, we show how to mount a key-recovery attack on PRINCE.

1. Query 2^{32} ciphertexts to PRINCE with the real key $k = (k_0, k_1)$, and obtain plaintext/ciphertext pairs denoted as (P_i, C_i). Store them in a hash table T_c indexed by $X_i = P_i \oplus C_i$.
2. Query 2^{32} plaintexts to PRINCE with the related key $k' = (k_0, k_1 \oplus \alpha)$, and obtain plaintext/ciphertext pairs denoted as (P'_i, C'_i). Store them in a table T_p index by $Y_i = P'_i \oplus C'_i$.
3. Find collisions in the keys of T_p and T_c.
4. For each pair $X_i = Y_j$, compute $Z = C_i \oplus P'_j$. Sample a plaintext P uniformly at random, and obtain the corresponding ciphertext C from the encryption oracle. Check the distinguisher by constructing the ciphertext $C' = P \oplus Z$, querying its corresponding plaintext P' decrypted with the related-key, and check if $P' \oplus C = Z$. If this holds, then $Z = k_0 \oplus L(k_0)$.
5. Retrieve k_0 by inverting the bijection $x \to L(x) \oplus x$, and finish the attack by recovering k_1 by exhaustive search.

Complexity analysis. After the two first steps where two structures of 2^{32} independent values have been constructed, by the birthday paradox we expect one collision for step 3. This collision gives a suggestion for k_0 that we check with the previously described distinguisher. This attack requires known-plaintexts, but we note that with a chosen-plaintext attack, we can pick C_i and P'_j carefully such that $C_i \oplus P'_j$ covers all the possible 2^{64} values. This ensures the value of k_0 to be recover with probability 1 at Step 4.

The total data complexity is about 2^{33} chosen-plaintexts to construct the two tables and check the distinguisher, and requires a time complexity equivalent to $2^{33} + 2^{64} \approx 2^{64}$ encryptions. We recall that the security bound for single-key attack with 2^{33} data claimed by the designers equals $127 - 33 = 94$ bits.

3.2 Related-key boomerang attack on $\texttt{PRINCE}_{core}$

In this section, we describe a related-key boomerang attack on $\texttt{PRINCE}_{core}$ with a time complexity equivalent to 2^{48} encryptions. To construct the boomerang distinguisher, we split the core block cipher E of PRINCE into two halves $E = E_1 \circ E_0$, where both E_0 and E_1 consists in 6 non-linear layers. The main observation that makes the distinguisher efficient is the existence of related-key differential characteristics with a very high probability. We start our analysis with an inspection of the S-box of PRINCE.

Property 2. For the S-box of PRINCE, there are 15 differential transitions, i.e. 15 pairs of input-output differences, that hold with probability 2^{-2}.

Further, we introduce three differences $(\Delta, \Delta_M, \nabla)$ that play the main role in our boomerang attacks. Let $\Delta \to \Delta_O$ be one of the 15 transitions with probability 2^{-2}, and let Δ_M be defined as $\Delta_M = M(\Delta_O)$, where M is the linear layer of PRINCE. Finally, let $\nabla = \Delta \oplus \Delta_M$.

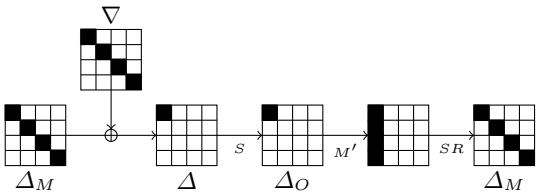

Fig. 3. Iterative differential characteristic on one round of $\texttt{PRINCE}_{core}$ used in the boomerang distinguisher.

Property 3. For $\texttt{PRINCE}_{core}$, there exists one round iterative characteristic $(\Delta_M, \nabla) \to (\Delta_M)$ where Δ_M is the difference in the incoming state and ∇ is the difference in the key, that holds with probability 2^{-2}.

The proof is trivial and is based on the particular values of the differences we have defined above (see Fig.3).

The related-key boomerang distinguisher uses two independent six-round differential characteristics, produced as concatenation of six copies on the single-round differential characteristic previously described. Thus, we obtain two six-round characteristics with probabilities $p = q = 2^{-2\times 6} = 2^{-12}$. Consequently, the related-key boomerang distinguisher finds a boomerang quartet of plaintexts in $(pq)^{-2} = 2^{48}$ queries to the encryption/decryption oracle. We have implemented the distinguisher on a PC and found out that due to the amplified probability of the boomerang, the actual complexity is lower, i.e. it is somewhere around 2^{36}. Thus, we were able to find a boomerang quartet for the full 12 rounds of $\texttt{PRINCE}_{core}$. An example of one such quartet is given in Appendix A.

Before we continue, we would like to make a few observations regarding the boomerang:

- the distinguisher is applicable regardless of the choice of the diffusion matrix M,
- the distinguisher is applicable regardless of the position of Δ in the state, i.e. we can choose any of the 16 nibbles,
- the distinguisher is applicable regardless of the choice of Δ in the top and the bottom characteristics,
- for one of the six-round characteristics one can choose differential transition that holds even with probability 2^{-3}. In that case, the probability of the boomerang becomes $2^{6\cdot 2\cdot(-3)+6\cdot 2\cdot(-2)} = 2^{-60}$.

Thus we can conclude that for $\texttt{PRINCE}_{core}$, one can launch around $15 \cdot 16 \cdot 15 \cdot 16 \approx 2^{16}$ different related-key boomerang distinguishers that hold with probability 2^{-48}, and around $210 \cdot 16 \cdot 15 \cdot 16 + 15 \cdot 16 \cdot 210 \cdot 16 \approx 2^{21}$ boomerangs with probability 2^{-60}. In the sequel, we denote $\mathcal{A}(i, j)$ the boomerang distinguisher with probability 2^{-48} where the active on the top characteristic is the i-th one, and the j-th one for the bottom characteristic, $0 \leq i, j < 16$.

Key-recovery attack. We now show how to turn the previous related-key boomerang distinguisher into a key-recovery attack. After the previously described distinguishing algorithm has completed, the attacker has one boomerang structure consisting in two pairs conforming to the first differential characteristic, and two other pairs verifying the second differential characteristic. From the plaintext, we show that the entropy of the nibble from k_1 corresponding to the active nibble in the top characteristic has been reduced to 2 bits. Indeed, as the pair verifies the first round, we know the differential transition of the first active nibble, so that there are only 4 possible values of that particular nibble[3]. Since we know the values in the plaintexts, and we have two pairs that verify this transition, the corresponding key-nibble can only take two values. The same reasoning applies on the ciphertexts for the bottom characteristic.

If we run 16 different instances of the boomerang distinguishing algorithm $\mathcal{A}(n, n)$, $0 \leq n < 16$, with the same nibble position n in the two characteristic, each iteration would narrow the n-th nibble of k_1 to exactly one value, but this would also require $16 \cdot 2^{36}$ chosen-plaintexts. Instead, we run 8 times the algorithm with different nibble positions in the top and the bottom part: $\mathcal{A}(n, n+8)$, $0 \leq n < 8$. Consequently, the information from the top pairs reduces the left half of k_1 to 2^8 values, and the bottom pairs reduces the right half of k_1 to 2^8 values as well. In total, this requires $8 \cdot 2^{36}$ data and time to run the boomerang algorithm $\mathcal{A}$, and an additional 2^{16} time to recover the actual key k_1.

4 A Single-key Attack on $\texttt{PRINCE}_{core}$ With Chosen α

The related-key boomerang attack presented above does not make use of the α-reflection property, but rather of the high probability one-round iterative characteristic. In this section, we show that the two concepts can be combined into a single-key boomerang attack with a modified value of α, i.e. we show existence of a set of values of $\alpha \neq 0$ for which one can launch key-recovery attack on $\texttt{PRINCE}_{core}$. The idea of our single-key attack is to align encryption with decryption in the boomerang. We note that the possibility of alignment has been discussed in the submission (see Sect. 3.1 of [4]), however the designers did not examine the case of boomerangs.

First, let us assume the encryption Enc of $\texttt{PRINCE}_{core}$ is aligned with decryption Dec, and focus on differential trails. Due to the α-reflection property, these two primitives are identical up the the addition of the round constants RC_i. As pointed by the designers, to build a related-key differential trail between Enc and Dec, one takes difference α in the related keys and since the same difference α is introduced by the round constants, in each round the differences cancel and the trail holds with probability 1. On the other hand in the single-key case, the difference coming from the key is 0, while the constants would still have the predefined α. Recall that in the six-round differential trails used in the related-key boomerang attack, in each round the difference introduced by the key is ∇. Hence, if α would coincide with the difference ∇ in the key from the above related-key boomerang, then a six-round single-key trail between Enc and Dec is precisely the same as the six-round related-key trail between two Enc (or between two Dec), i.e. the keys and constants switch roles. In other words, in the single-key case one can build a six-round trail with probability 2^{-12}.

The single-key boomerang attack for the whole $\texttt{PRINCE}_{core}$ uses the same Δ_M in the top and bottom characteristics, and it can be described as follows:

1. **Aligning encryption with decryption at the beginning:** Take a random plaintext P_1 and compute $C_2 = P_1 \oplus \Delta_M$.
2. **Aligning two encryptions with decryptions at the end:** Encrypt P_1 to produce the ciphertext C_1, and decrypt C_2 to produce the plaintext P_2. Compute $C_3 = C_1 \oplus \Delta_M$ and $P_4 = P_2 \oplus \Delta_M$.
3. **Aligning encryption with decryption at the beginning:** Decrypt C_3 to produce the plaintext P_3. Encrypt P_4 to produce the ciphertext C_4. If $P_3 \oplus C_4 = \Delta_M$ output the boomerang quartet (P_1, C_2, P_3, C_4), otherwise go to step 1.

[3] The transitions occurring with probability 2^{-2}, there are two pairs of values that are solution to $S(x) \oplus S(x \oplus \Delta) = \Delta_O$.

After repeating 1-3 around 2^{48} times, one finds the quartet with a high probability. The proof of correctness of the above boomerang is similar as in the case of standard boomerangs (where one aligns encryptions with encryptions).

In the single-key case, we cannot choose the position of the active nibble as it is fixed by the value of α. Thus in the key recovery attack, we can recover only a single nibble of the master key. The first boomerang quartet will suggest 4 possible values for this nibble, and an additional quartet will give the exact value. Thus the complexity of recovering 4 bits of the master key is $2 \cdot 2^{48} = 2^{49}$. The remaining 60 bits can be searched exhaustively. Our experimental results suggest that when the top and the bottom characteristic use the same value Δ_M then the probability of the boomerang is somewhat lower, i.e. instead of 2^{-36} obtained in the case of different Δ_M, now we get 2^{-40}. Therefore the actual recovery of the 4 bits is around $2 \cdot 2^{40} = 2^{41}$.

The above attack is applicable only when the value of the constant α coincides with the value of Δ_M defined in the previous section. Therefore, α can take $15 \cdot 16 = 240$ different values. We note that the original value chosen by the designers is not among these 240 values.

5 Exploiting the extra linear relation

In this section, we give an analysis of PRINCE in the single-key model. We show that while the claim of the authors is that no attack can be conducted on PRINCE with less than 2^{127-n} computations with 2^n queries, it is possible to slightly break this bound by leveraging the various linear relations that exist with probability 1 in the cipher. Of course, considering the small gain factor (only about $2^{0.6}$), our attack does not really contradict the claim of the designers. However, it indicates that perhaps it might be possible to tweak the security proof in order to take in account all the linear relations inherent to the structure of PRINCE. We emphasize that the gain factor comes directly from the number of keys tested, not by computing only parts of the cipher as for biclique attacks [3]. It would be possible to slightly increase the gain by combining with the accelerating tricks from biclique attacks, but our goal is not in this direction as we are analyzing the structural behavior of the cipher.

5.1 The linear relations

The idea underlying our attack is that there exist two linear relations for PRINCE cipher that are verified with probability 1:

$$E_{(k_0||k_1)}(P) = E_{(k_0 \oplus \Delta||k_1)}(P \oplus \Delta) \oplus L(\Delta) \quad \text{or} \quad D_{(k_0||k_1)}(C) = D_{(k_0 \oplus \Delta||k_1)}(C \oplus L(\Delta)) \oplus \Delta \tag{2}$$

$$D_{(k_0||k_1)}(C) = E_{(k_0||k_1 \oplus \alpha)}(C \oplus k_0 \oplus L(k_0)) \oplus k_0 \oplus L(k_0) \tag{3}$$

The first equation (2) is the simple related-key relation due to the Even-Mansour construction of PRINCE, while the second equation (3) is the α relation required for the smooth decryption of PRINCE. Using these two relations, we will be able to test 4 keys at the same time, with only one PRINCE computation, thus leading to a maximal gain factor of 2 over the claimed security (2^{127} with a single query).

First let us assume that we queried some plaintext P to the encryption oracle and we received ciphertext C. By picking a random key $(k_0||k_1)$, the attacker can compute $E_{(k_0||k_1)}(P) = C'$ and directly check if $C' = C$. If not, then he knows that $(k_0||k_1)$ is not the secret key. However, he can deduce more than just this information. Indeed, from (2) and by denoting $C' \oplus C = \delta \neq 0$, we deduce

$$\begin{aligned} E_{(k_0 \oplus L^{-1}(\delta)||k_1)}(P \oplus L^{-1}(\delta)) &= E_{(k_0||k_1)}(P) \oplus L(L^{-1}(\delta)) \\ &= C' \oplus \delta = C \end{aligned}$$

and since $\delta \neq 0$, then $L^{-1}(\delta) \neq 0$ and thus the key $(k_0 \oplus L^{-1}(\delta)||k_1)$ encrypts a different plaintext than P to ciphertext C, i.e. it is not a valid key (and it is different from key $(k_0||k_1)$ since $L^{-1}(\delta) \neq 0$).

At this point, the attacker can test two keys with one PRINCE query and one PRINCE offline computation. However, he can deduce even more information by using equation (3) and using notation $X = L^{-1}(P \oplus C \oplus k_0)$:

$$\begin{aligned} D_{(X||k_1 \oplus \alpha)}(C) &= E_{(X||k_1)}(C \oplus X \oplus L(X)) \oplus X \oplus L(X) \\ &= E_{(k_0||k_1)}(C \oplus X \oplus L(X) \oplus k_0 \oplus X) \oplus X \oplus L(X) \oplus L(k_0 \oplus X) \\ &= E_{(k_0||k_1)}(P) \oplus L(k_0) \oplus X \\ &= C' \oplus L(k_0) \oplus L^{-1}(P \oplus C \oplus k_0) \end{aligned}$$

and if $C' \oplus L(k_0) \oplus L^{-1}(P \oplus C \oplus k_0) \neq P$, then it means that the key $(X||k_1 \oplus \alpha)$ deciphers the ciphertext C to a plaintext different from P, i.e. it is not a valid key. Moreover, using notation $Y = P \oplus C' \oplus L(k_0)$, we can

also write:

$$\begin{aligned}
E_{(Y||k_1\oplus\alpha)}(P) &= D_{(Y||k_1)}(P \oplus Y \oplus L(Y)) \oplus Y \oplus L(Y) \\
&= D_{(k_0||k_1)}(P \oplus Y \oplus L(Y) \oplus L(k_0 \oplus Y)) \oplus Y \oplus L(Y) \oplus k_0 \oplus Y \\
&= D_{(k_0||k_1)}(C') \oplus k_0 \oplus L(Y) \\
&= P \oplus k_0 \oplus L(P \oplus C' \oplus L(k_0))
\end{aligned}$$

and if $P \oplus k_0 \oplus L(P \oplus C' \oplus L(k_0)) \neq C$, then it means that the key $(Y||k_1 \oplus \alpha)$ ciphers the plaintext P to a ciphertext different from C, i.e. it is not a valid key.

5.2 Speeding up the key recovery with linear relations

For previous subsection, it is clear that with only a single query to the encryption oracle, and performing only a single PRINCE offline computation, one can eliminate four keys at a time (namely $K_1 = (k_0||k_1)$, $K_2 = (k_0 \oplus L^{-1}(\delta)||k_1)$, $K_3 = (L^{-1}(P \oplus C \oplus k_0)||k_1 \oplus \alpha)$ and $K_4 = (P \oplus C \oplus \delta \oplus L(k_0)||k_1 \oplus \alpha))$ by testing simple linear relations. However, there is a subtlety here because among the four keys that are tested, some are uncontrolled by the attacker. Indeed, while K_1 is directly chosen by the attacker, the value of the tested keys K_2 or K_4 depend on δ which is a random value from the view of the attacker. The third key K_3 does not depend on δ and therefore can be chosen by the attacker as well (that is, k_0 and k_1 linearly define K_1 and K_3).

We would like to evaluate the complexity of a brute force key search using this method that tests four keys with only a single PRINCE computation. One can first divide the sets of keys k_1 into 2^{63} independent pairs $(k_1, k_1 \oplus \alpha)$. The attacker will go through the 2^{63} pairs and for each of them test all the possible values of k_0. For each PRINCE computation, he will eliminate two keys for k_1 (i.e. K_1 and K_2) and two keys for $k_1 \oplus \alpha$ (i.e. K_3 and K_4), continuing until he has tested all the keys k_0 for both k_1 and $k_1 \oplus \alpha$, and then going to the next pair $(k_1, k_1 \oplus \alpha)$. To minimize the overall complexity, at each step the attacker will select a value for k_0 such that key K_1 and key K_3 have not been tested yet and this can be done with a good probability[4] as long as the number of untested keys k_0 for both k_1 and $k_1 \oplus \alpha$ is bigger than 2^{32}. The two others keys K_2 and K_4 will randomly hit either a new and untested key or an already tested one, but on average over the whole process about one key will be eliminated. Overall, with one PRINCE computation on average about three new key candidates are removed and the total key recovery complexity is about $2^{128}/3 = 2^{126.4}$ PRINCE evaluations, while with a single query to the encryption oracle the security claim by the designers is 2^{127}. We give in Appendix B a slightly more precise analysis of the attack complexity, leading to $2^{126.47}$ computations.

5.3 Generalization to several queries

In the previous subsection, only a single plaintext query was sent to the encryption oracle, but in fact this is not enough to fully recover the PRINCE secret key since at least two 64-bit queries are required to fully determine the 128-bit secret key. Asking one more query to the oracle in order to prune the remaining key candidates will reduce by a factor 2 the security claim given by the designers which will become lower than our key recovery complexity. Therefore, we need to generalize our previous attack to the case of several oracle queries, and we analyze the example of two queries.

Our goal with two queries is now to be able to test 8 keys at a time (instead of 4), using only one offline PRINCE computation. Let us assume that in addition to the first query (P, C), we also ask for the encryption of $P \oplus 1$ and we receive C^{+1}. As before, by choosing a random key $(k_0||k_1)$ and computing offline $E_{(k_0||k_1)}(P) = C'$, we can test four keys at a time by using (P, C). It is actually straightforward to apply the very same reasoning to $(P \oplus 1, C^{+1})$ as well and get to test four more keys for free. For example, similarly to the first key K_1 we can write:

$$\begin{aligned}
E_{(k_0\oplus 1||k_1)}(P \oplus 1) &= E_{(k_0||k_1)}(P) \oplus L(1) \\
&= C' \oplus L(1)
\end{aligned}$$

and if $C' \oplus L(1) \neq C^{+1}$, then it means that the key $(k_0 \oplus 1||k_1)$ ciphers the plaintext $P \oplus 1$ to a ciphertext different from C^{+1}, i.e. it is not a valid key. We can apply this kind of transformation to the three other keys K_2, K_3, K_4 and obtain three more free keys.

During the key recovery process, we now get a structure with 8 tested keys, where 4 are for k_1 (two controlled and two uncontrolled) and 4 are for $k_1 \oplus \alpha$ (two controlled and two uncontrolled). With the very same reasoning

[4] Since there are 2^{64} values of k_0 to test, there will always be at least 2^{32} untested key for both k_1 and $k_1 \oplus \alpha$ except at the very end of the process, but then the effect is negligible since only 2^{32} keys will remain to be tested.

as before[5], we deduce that 6 new keys are tested on average per offline PRINCE computation, and the final key recovery complexity is $2^{128}/6 = 2^{125.4}$ PRINCE evaluations, while with two queries to the encryption oracle the security claim by the designers is 2^{126}. Using the same reasoning than depicted in Appendix B, we obtain a slightly more precise analysis of the attack complexity, leading to $2^{125.47}$ computations.

6 Integral attacks for reduced-round $\texttt{PRINCE}_{core}$ and PRINCE

In this section, we present key-recovery attacks for reduced variants of 4, 5 and 6 rounds of $\texttt{PRINCE}_{core}$, and show how to extend them to key-recovery attack on the same number of rounds for PRINCE. The basic strategy comes as a direct application of the SQUARE attack proposed in [5]. We begin by describing the context for PRINCE with a 4-round version, and then show how to extend it to 5 and 6 rounds. In the sequel, we use the notations defined in Section 2 where the middle layer M_{mid} is linear.

6.1 Attack on 4 rounds

This small version considers two rounds $\mathcal{R}_0$ and $\mathcal{R}_1$ in the first part of the core block cipher, followed by the middle linear layer M_{mid}, and finally the two last rounds $\mathcal{R}_2$ and $\mathcal{R}_3$. The secret key to recover for $\texttt{PRINCE}_{core}$ is k_1. This attack, as well as the subsequent ones, uses the following 3-round distinguishing property as its core.

Property 4. Let $\mathcal{P}_n$ be a set of 2^4 plaintexts such that a particular nibble n assumes all 2^4 possible values while the 15 other ones are fixed to chosen constants. We call this structure a δ-set. The encryption of the δ-set $\mathcal{P}_n$ through three rounds of $\texttt{PRINCE}_{core}$ produces a set $\mathcal{C}$ where all nibbles are balanced, that is:

$$\forall n \in \{0, \ldots, 15\}, \bigoplus_{c \in \mathcal{C}} c[n] = 0.$$

The proof strictly follows the one from [5] and is due to the wide-trail strategy followed by the designers. Additionally, we can also consider the encryption of $\mathcal{P}_n$ under 3.5 rounds of $\texttt{PRINCE}_{core}$, where we skip the application of the non-linear layer in the fourth round. Applying the S-box destroys this algebraic property of the δ-set, but allows to mount a key-recovery attack.

We begin by constructing a δ-set $\mathcal{P}_0$ of 2^4 plaintexts where nibble at position 0 assumes all 2^4 values, and we ask the encryption $\mathcal{P}_0$ under the secret key k_1 and store the ciphertexts in $\mathcal{C}$. Then, for all nibbles n in k_1, guess the value of $k_1[n]$ and compute $\sigma = \bigoplus_{c \in \mathcal{C}} S\Big(c[n] \oplus k_1[n] \oplus RC_4[n]\Big)$. If $\sigma = 0$, then the nibble before the last non-linear layer is balanced, and we get a valid suggestion for the value $k_1[n]$. Otherwise, we discard the guess.

This algorithm requires 2^4 chosen plaintexts and suggests in average one value per nibble of k_1 since each check should remove 1 out of 2^4 guesses. At the end, we recover in sequence all the nibbles of k_1 with a total time complexity of $16 \cdot 2^4 = 2^8$ simple operations, and 2^4 64-bit words of memory.

6.2 Attack on 5 rounds

Further we show how to add one round at the end of the previous attack, to reach five rounds. We note that this reduced variant of $\texttt{PRINCE}_{core}$ is not symmetric since there are two rounds, $\mathcal{R}_0$ and $\mathcal{R}_1$, before M_{mid} and three rounds after: $\mathcal{R}_2$, $\mathcal{R}_3$ and $\mathcal{R}_4$. The strategy remains the same: we guess particular key nibbles to check the distinguishing property on an encrypted δ-set $\mathcal{C}$. Now we need to guess 4 nibbles of a column of k_1 to partially decrypt the corresponding columns of the ciphertexts and check the balanced property. Note that in the case of $\texttt{PRINCE}_{core}$, we only need to guess 4 nibbles since there is no key-schedule, whereas for the AES we would need 5.

In comparison to the previous attack where one check suffices to remove all but one key guess, here we need more. Indeed, we expect a single check to behave as a 4-bit filter, so that 4 δ-sets should provide enough information to discard all but 1 key guess. In practice, we measure that the filter is not that strong: we require in average 4.7 δ-set to determine the 4 key nibbles uniquely. In total, the attack requires $5 \cdot 2^4$ chosen plaintexts, $5 \cdot 2^4$ memory to store them, and a time complexity of $4 \cdot 5 \cdot 2^{16} \approx 2^{21}$ simple operations to recover the full k_1.

[5] We have 4 controlled keys, which can be chosen to be always untested keys as long as the number of untested keys k_0 for both k_1 and $k_1 \oplus \alpha$ is bigger than 2^{48}. Since there are 2^{64} values of k_0 to test, there will always be at least 2^{48} untested key for both k_1 and $k_1 \oplus \alpha$ except at the very end of the process, but then the effect is negligible since only 2^{48} keys will remain to be tested.

6.3 Attack on 6 rounds

On top on the previous attack, we add one additional round at the beginning to reach six rounds. The strategy is the same as the one for the AES: we construct a set of plaintexts $\mathcal{P}$ such that we can construct a δ-set after one round. To do so, we consider a larger structure of 2^{16} plaintexts where the four diagonal nibbles assume all the possible values, and we ask its encryption to get the set of corresponding ciphertexts $\mathcal{C}$. Then, we guess the four diagonal nibbles of k_1 and partially encrypt the associated data under the key guess to find 2^4 plaintexts/ciphertexts pairs defining a δ-set in the second round. We expect 2^{12} δ-sets $\mathcal{P}_i$ for any nibble i, so the data can be reused to replay the attack on a different δ-set. We can now apply the 5-round attack by guessing only 3 additional nibbles: we already know one in each column from the diagonal guess. In total, the attack requires 2^{16} chosen plaintexts of data and same for memory requirements and runs in time equivalent to $4 \cdot 2^{16} \cdot 2^{12} = 2^{30}$ simple operations.

6.4 Extension from $\texttt{PRINCE}_{core}$ to PRINCE

All the three previous attacks on $\texttt{PRINCE}_{core}$ can be extended to attacks on PRINCE by guessing the same nibbles in $L(k_0)$. Namely, if we have an integral attack on r rounds of $\texttt{PRINCE}_{core}$ requiring g precise guesses in the last application k_1, we can deduce an attack recovering $k_1 \oplus L(k_0)$ on the same number r of rounds by guessing the same g nibbles in both k_1 and $L(k_0)$. For each correct final guess g that verifies the balanced property, we deduce the right value for $k_1[g] \oplus L(k_0)[g]$. Hence, for the 6-round attack, we can recover $k_1 \oplus L(k_0)$ with 2^{16} chosen plaintexts and $(2^4)^{4+3+4} = 2^{44}$ simple operations. We first guess the four diagonal nibbles of k_1 to find the δ-set, then we guess 4 nibbles in a column of $L(k_0)$ and three new guesses in the same column of k_1 to partially decrypt the ciphertexts. For the same reason as before, only three guesses are needed in k_1 because we already know one. Finally, we can exhaust the 2^{64} values of either k_0 or k_1 to recover the full 128-bit master key.

7 Time-Memory-Data Tradeoffs

In this section, we present tradeoffs for the construction used in PRINCE, i.e. our approaches work regardless of the cipher used as $\texttt{PRINCE}_{core}$. The proposed tradeoffs are based on a property that the cipher can be divided into two parts, leading to a similar division of the phases of the key recovery attack. Then, one side of the attack is precomputed as it does not depend on the plaintext-ciphertext, while the other side is data-dependent and it is recomputed in the online phase. Depending on the precomputation phase and in particular on the memory used in this phase, our tradeoffs are based either on the meet-in-the-middle (MITM) attacks or on Hellman's tradeoffs[9]. We note that we give time-memory-*data* tradeoffs, i.e. we show that one can achieve tradeoffs involving data as well. This is not the case for the rest of the block ciphers, as the only known generic block cipher tradeoff is the Hellman's tradeoff which does not make use of larger data set.

We assume the reader is familiar with the Hellman's time-memory tradeoff that consists of two phases: 1) precomputation or offline phase, when the attacker encrypts a chosen plaintext under all possible keys and stores part of the results in so-called Hellman's tables, and 2) online phase, in which the attacker recovers the secret key using the tables. A cryptanalytic tradeoff is defined by the following parameters:

- N is the size of the key space (e.g. for PRINCE $N = 2^{128}$)
- P is the time complexity of the precomputation phase
- M is the amount of the memory used in both the precomputation and the online phases
- T is the time required to recover the secret key, i.e. the complexity of the online phase
- D is the amount of data required to recover the secret key

The standard way of presenting a tradeoff is by giving its curve, which is a simple relation between the time, memory, data, and the size of the key. The Hellman's time-memory tradeoff is the only known generic tradeoff for block ciphers, and has the curve $TM^2 = N^2, M > N^{\frac{1}{2}}$ and $P = N$. We use (P, C) to denote the plaintext-ciphertext pair for PRINCE, and (A, B) to denote the pair for $\texttt{PRINCE}_{core}$.

Our tradeoffs exploit the linearity of the addition of k_0. Recall that the addition of the key k_0 is defined as:

$$P \oplus k_0 = A \tag{4}$$

$$B \oplus L(k_0) = C, \tag{5}$$

or equivalently

$$L(P) \oplus L(A) = L(k_0) \tag{6}$$

$$B \oplus C = L(k_0). \tag{7}$$

Thus, the values of P, C, A, B are related as:

$$L(P) \oplus C = L(A) \oplus B \tag{8}$$

Therefore, the separation of (P, C) on one side, and (A, B) on the other is manageable. We note that a similar reduction was presented in [7]. It was applied to the case of single-key Even-Mansour, where $L(k_0) = k_0$, and the inner transformation F is a permutation rather than a cipher as in our case. However, [7] does not examine the possibility of tradeoff attacks.

A MITM Tradeoff. Our first tradeoff is MITM based. It can be described as follows:

1. In the precomputation phase, the attacker fixes 2^{64-d} values of A and for all possible 2^{64} values of the key k_1 computes the corresponding value of $B = PRINCE_{core}(A, k_1)$ and stores the tuple $(L(A) \oplus B, A, B, k_1)$ in a table $\tilde{S}$. The size of $\tilde{S}$ is 2^{128-d}.
2. In the online phase, given 2^d pairs of known plaintexts-ciphertexts, for each pair (P_i, C_i), the attacker computes the value of $L(P) \oplus C$ and checks for a match in the table $\tilde{S}$. For every found match, the outer key k_0 is computed, and a possible candidate $k_0||k_1$ is checked on a few more pairs of plaintexts-ciphertexts.

As there is 2^d data, the size of the set $\tilde{S}$ is 2^{128-d}, and the matching space is only 64 bits, there would be $2^{d+128-d-64} = 2^{64}$ candidates, thus the correct key would be found with an overwhelming probability.

This tradeoff has the following parameters:

$$N = 2^{128}, P = 2^{128-d}, M = 2^{128-d}, T = 2^{64}, D = 2^d, \tag{9}$$

and thus the precomputation phase is smaller than N, i.e. $PD = N$, while the resulting memory-data tradeoff curve is of the type:

$$DM = N, T = N^{\frac{1}{2}}, M > N^{\frac{1}{2}}. \tag{10}$$

Interestingly, this is precisely the curve given by Babbage and Golić [1, 8] for stream ciphers. Compared to the Hellman's curve, we get $TM^2 = 2^{64}2^{4\cdot 64-2d} = 2^{4\cdot 64}2^{64-2d} = N^2 2^{64-2d}$, hence when the data $D > N^{\frac{1}{4}} = 2^{32}$, we get a better tradeoff.

Hellman's tables trade-off. Though the time complexity seems attractive as it is only $N^{\frac{1}{2}}$, the memory complexity required by the previous tradeoff is quite large. Hence, it is reasonable to try to reduce the memory by increasing the time. This is achievable by implementing Hellman's tradeoff as intermediate step of the tradeoff for the whole cipher. Hellman's tradeoff satisfies the curve $TM^2 = N^2$, where $N = 2^n$, $T = t^2, M = mt$, and $mt^2 = 2^n$. The values t, m are related to the dimension and the number of the tables created during the offline phase. Note that Hellman's tables are computed for a particular plaintext. We call *P-Hellman's tables*, the precomputation phase computed under the plaintext P. Thus P-Hellman's tables can recover the secret key if the supplied plaintext is P.

Our tradeoff based on Hellman's tables can be described as:

1. In the precomputation phase, the attacker creates a set $\tilde{S}$ of 2^{n-d} different values A_i for A and for each value, builds A_i-Hellman's tables for the cipher $PRINCE_{core}(A_i, k_1)$.
2. In the online phase, given 2^d pairs of known plaintexts-ciphertexts, for each pair (P_i, C_i), the attacker performs the following steps:
   - Fixes one value of A_i from the predefined set $\tilde{S}$,
   - Computes the value of $k_0 = P_i \oplus A$,
   - Computes the corresponding value of $B = C_i \oplus L(k_0)$,
   - Uses A_i-Hellman's table, to find a value of k_1 such that $PRINCE_{core}(A_i, k_1) = B$,
   - Checks if the found key $k_0||k_1$ is the correct key by testing on a few more pairs of plaintext-ciphertext,
   - If the suggested key is incorrect, repeats all of the above steps.

As there is 2^d data, and 2^{64-d} values of A_i in $\tilde{S}$, in total there are $2^d 2^{64-d} = 2^{64}$ possible values for the key k_0, and for each of them on average one value for the key k_1, or 2^{64} pairs of suggested keys, thus the attacker finds the right key with a high probability. In the precomputation phase, for a single value of A, the attacker uses 2^{64} computations to build Hellman's tables and requires $M = mt$ memory to store each of them. In the online phase, given A and B, the attacker needs $T = t^2$ time to find the correct value of the key k_1. Therefore, the tradeoff has the following parameters:

$$N = 2^{128}, P = 2^{128-d}, M = 2^{64-d}mt, T = 2^{64}t^2, D = 2^d, \tag{11}$$

and the resulting time-memory-data tradeoff curve is of the type:

$$T(MD)^2 = 2^{64}t^2 2^{2\cdot 64-2d}m^2t^2 2^{2d} = 2^{3\cdot 64}(t^2m^2t^2) = 2^{3\cdot 64}2^{2\cdot 64} = 2^{5\cdot 64} = N^2 N^{\frac{1}{2}}. \tag{12}$$

Again, our tradeoff compared to the Hellman's tradeoff is better at the points of the curve where $D > N^{\frac{1}{4}}$. We should note that due to the claimed security level of `PRINCE`, i.e. $TD < N$, an additional requirement $M^2D > 2^{192}$ is introduced.

Hellman's single table trade-off. In the Hellman's tradeoff, different tables, each with a unique reduction function, are created in order to avoid colliding chains, i.e. if the chains are too long, the probability they will collide is high and therefore either the precomputation time has to be increased or the number of keys that can be recovered in the online phase becomes small. The collisions in the precomputation phase cannot be detected, hence the chains are kept short. However, the situation changes if one can store all of the values. This type of scenario is discarded in the classical Hellman's tradeoff as it requires $M = N$. However, in the case of $\texttt{PRINCE}_{core}$, the required memory is only $M = N^{\frac{1}{2}}$ which is precisely the lower bound on the memory in the Hellman's tradeoff (recall that the memory requirement in the Hellman's tradeoff is $M > N^{\frac{1}{2}} = 2^{64}$). Using 2^{64} memory, one can easily create a single Hellman's table for the whole tradeoff – the table has m chains, each with around t points. The first chain starts with a terminal point (a value that does not have a preimage) and can have a length of up to 2^{32}, i.e. $t < 2^{32}$. If the length t is chosen to be less than 2^{32}, then the starting point of the next chain is the end point of the previous one. This process is repeated until a collision is obtained – such collision can be detected immediately as one has all the values stored. Once a collision occurs, the next chain starts again with a terminal point. Hence, to build the whole table, one needs 2^{64} time and memory, and $mt = 2^{64}$. Only the starting and end points of the chains are stored for the online phase, thus the memory of the online phase is m, while the time complexity is t, and therefore the tradeoff curve becomes $TM = N$. Note that the memory 2^{64} is reusable across different tables, i.e. if one wants to create different tables for tradeoffs with different plaintexts, the same 2^{64} can be used. Also, as the chains can have a maximal length of 2^{32}, if follows that $t \leq 2^{32}$ and $m \geq 2^{32}$.

The tradeoff presented above can be tweaked, and instead of building multiple Hellman's tables with $mt^2 = 2^{128}$, we can use the single table described here with $mt = 2^{64}$. Hence, using this technique, we obtain the following tradeoff:

$$N = 2^{128}, P = 2^{128-d}, M = max(2^{64-d}m, 2^{64}), T = 2^{64}t, D = 2^d, \tag{13}$$

and the resulting time-memory-data tradeoff curve is of the type:

$$TMD = 2^{64}t2^{64-d}m2^d = 2^{2\cdot 64}(tm) = 2^{2\cdot 64}2^{64} = NN^{\frac{1}{2}}. \tag{14}$$

Obviously $M > N^{\frac{1}{2}}$ has to hold (same as in the Hellman's tradeoff), but now we get that for any $D > M/N^{\frac{1}{2}}$ our tradeoff is better than Hellman's, that is if one uses 2^{64+d} memory, and can obtain more than 2^d known pairs of plaintext-ciphertext, by implementing our tradeoff he can recover the key with less computations then by implementing the generic Hellman's tradeoff. We emphasize that our tradeoff requires only known data, i.e. it is far more practical requirement, than the one of the generic tradeoff.

Acknowledgement

The authors would like to thank the FSE 2013 reviewers and the Prince team for their valuable comments. Ivica Nikolić is supported by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03. Thomas Peyrin, Lei Wang and Shuang Wu are supported by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06.

References

1. Steve Babbage. A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers. 1995. European Convention on Security and Detection, IEE Conference Publication No. 408.
2. Alex Biryukov. DES-X (or DESX). In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, page 331. Springer, 2011.
3. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
4. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzi Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Soren S. Thomsen, and Tolga Yalçın. PRINCE: A Low-latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT. to appear*, 2012.
5. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.
6. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer, 2002.
7. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2012.
8. Jovan Dj. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.

9. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
10. Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *J. Cryptology*, 14(1):17–35, 2001.

A Example of a boomerang structure

We present here an example of a boomerang structure found for the attack described in Section 3.2.

Table 2. Example of a related-key boomerang structure $\Big((k_i, p_i, c_i)\Big)_{i=1,\ldots,4}$ for the full $\texttt{PRINCE}_{core}$ in hexadecimal values.

$(k_1, k_2, k_1 \oplus k_2)$	91b4e89d2625f1fb	91b5e88d2725f1fa	0001001001000001
$(p_1, p_2, p_1 \oplus p_2)$	0b92a736c9bb91a3	0b93a726c8bb91a3	0001001001000000
$(c_1, c_2, c_1 \oplus c_2)$	2f04603451d1d3df	3846bd541167b633	1742dd6040b665ec
$(k_3, k_4, k_3 \oplus k_4)$	91a4e99d2635f1fa	91a5e98d2735f1fb	0001001001000001
$(p_3, p_4, p_3 \oplus p_4)$	a763296ea531a6b8	a762297ea431a6b8	0001001001000000
$(c_3, c_4, c_3 \oplus c_4)$	2f14613451d1d3de	3856bc541167b632	1742dd6040b665ec
$(k_1, k_3, k_1 \oplus k_3)$	91b4e89d2625f1fb	91a4e99d2635f1fa	0010010000100001
$(p_1, p_3, p_1 \oplus p_3)$	0b92a736c9bb91a3	a763296ea531a6b8	acf18e586c8a371b
$(c_1, c_3, c_1 \oplus c_3)$	2f04603451d1d3df	2f14613451d1d3de	0010010000000001
$(k_2, k_4, k_2 \oplus k_4)$	91b5e88d2725f1fa	91a5e98d2735f1fb	0010010000100001
$(p_2, p_4, p_2 \oplus p_4)$	0b93a726c8bb91a3	a762297ea431a6b8	acf18e586c8a371b
$(c_2, c_4, c_2 \oplus c_4)$	3846bd541167b633	3856bc541167b632	0010010000000001

B Analysis of the key recovery attack complexity of Section 5

In the cryptanalysis described in Section 5, the attacker would like to test the entire set of the 2^k possible keys. At each step, four keys will be tested directly. However, for each step, the attacker can only choose the value of two keys, and the two others are randomly chosen among the set of all possible keys (thus potentially already tested ones). Since the overall complexity of the attack is the number of steps required to test the entire set of keys, we would like to evaluate this quantity precisely.

In order to ease the modeling, we consider the problem where at each step one key is chosen by the attacker (thus always an untested one) and another one is chosen randomly. Let $T_{1/2}$ be the step where half of the keys have already been tested. After $T_{1/2}$, at least one new key will be tested on average, since the attacker can choose one key each step. Before $T_{1/2}$, at least 1.5 new key will be tested on average, since the attacker can choose one key each step and since the randomly chosen key will have a probability greater than 1/2 to be an untested key. We can conclude that the average number of keys tested per step is at least $2/(1 + 1/1.5) = 1.2$.

We further continue the partitioning by denoting $T_{i/x}$ the step where a proportion i/x of all keys have already been tested. Then, with the same reasoning, after $T_{i/x}$ at least $(2 - (i + 1)/x)$ new keys will be tested on average and before $T_{i/x}$ at least $(2 - i/x)$ new keys will be tested on average. The approximation gets more precise as x grows and we obtain that the average number of key tested per step is equal to

$$\lim_{x\to\infty} \frac{x}{\sum_{i=0}^{x-1}(1/(1 + i/x))} = \frac{1}{\ln(2)} \approx 1.443. \tag{15}$$

As a consequence, the average number of steps required to test the entire key space in Section 5 is approximately $2^k/(2 \times 1.443) = 2^{k-1.53}$.