FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

David Tvrdý

Cryptanalytic attacks on the cipher PRINCE

Department of Algebra

Supervisor of the master thesis:	Dr. rer. nat. Faruk Göloglu
Study programme:	Mathematics
Study branch:	Mathematics for Information Technologies

Prague 2022

This thesis is dedicated to all the people who made this work possible. I am especially thankful to my supervisor Faruk Göloglu, under whose guidance I have completed this thesis, for his his valuable advice.

Title: Cryptanalytic attacks on the cipher PRINCE

Author: David Tvrdý

Department: Department of Algebra

Supervisor: Dr. rer. nat. Faruk Göloglu, Department of Algebra

Abstract: This work surveys and studies the most practical attacks on round reduced versions of the cipher PRINCE. Specifically, concepts of integral cryptanalysis and meet-in-the-middle attacks are discussed. A new 4.5-round integral distinguisher with lower time and data complexity is presented. A new meet-in-the-middle attack on seven rounds of the cipher with low data complexity is proposed. A Python 3 reference implementation of the cipher as well as of some of the integral attacks is provided.

Keywords: lightweight cryptography, PRINCE, cryptanalysis

Contents

Introduction **2**

1 Introduction to symmetric block ciphers **4**

1.1 Construction of symmetric block ciphers 4

1.1.1 Substitution-permutation network 4

1.1.2 FX construction 7

1.2 Attacks on symmetric block ciphers 7

1.2.1 Integral cryptanalysis 8

1.2.2 Meet-in-the-middle attacks 8

2 PRINCE cipher description **11**

2.1 Overview 11

2.1.1 The key schedule 11

2.2 PRINCE$_{\text{core}}$ 11

2.2.1 The linear layer 12

2.2.2 The alpha-reflection property 14

2.3 Round reduced versions 14

3 Integral attacks on PRINCE **15**

3.1 3.5-round integral distinguisher 15

3.2 4.5-round integral distinguishers 21

3.2.1 The original 4.5-round integral distinguisher 21

3.2.2 The new 4.5-round integral distinguisher 23

3.3 Faster key recovery technique 24

3.4 6-round integral attack 26

4 Meet-in-the-middle attacks on PRINCE **29**

4.1 6-round meet-in-the-middle attack 29

4.2 8-round meet-in-the-middle attack 33

4.3 7-round meet-in-the-middle attack 37

Conclusion **43**

Bibliography **44**

List of Figures **45**

A Attachments **47**

A.1 Attachment one - PRINCE Python 3 implementation 47

A.2 Attachment two - Python 3 implementation of round reduced integral attacks 47

A.3 Attachment three - Recovery of the remaining key bits after the 6-round meet-in-the-middle attack 48

A.4 Attachment four - Recovery of the remaining key bits after the 8-round meet-in-the-middle attack 49

Introduction

Cryptography is a study of methods of secure communication in the presence of an adversary. Its original goal was to construct and analyze algorithms and protocols which would prevent private messages from being exposed to the public. Though the original goal prevails, there are many other applications of cryptography in addition to secure communication, for example computer passwords and digital currencies.

Modern cryptography is based on both mathematical theory and computer science practice. Today's cryptographic algorithms are designed using computational hardness assumptions. Though it is theoretically possible to brake such algorithms, it is very hard, typically infeasible, in actual practice to do so for any adversary. Information-theoretically secure algorithms, which cannot be broken even with unlimited computational power, exist, however, they are very difficult to use in practice. Of course, advances in mathematical theory or computer technology may cause today's designs to be insecure in the future. That is why cryptographic algorithms have to be continuously analyzed and updated.

Recently, devices with constrained computational power have become very popular, especially in connection to the Internet of Things systems. Since such devices can also be targets of cyberattacks, it is important to protect their data by encryption as well. For example, imagine a situation where a large number of sensors with limited power is installed to collect real world data. The data is sent to a server which analyzes them, and, based on the analysis, it autonomously controls complex devices or technological systems, for example in a factory. If an attack would falsify the collected data, results of the analysis would be incorrect and suboptimal control would be performed, leading potentially to higher costs or damage. Lightweight cryptography is a branch of cryptography where the goal of the cryptographic algorithms is to achieve very low computational complexity and very low hardware requirements.

Although numerous lightweight ciphers have been proposed, none of them has been chosen as a standard so far. Nevertheless, studying these ciphers is still a valuable contribution to the research in the field of cryptography. New concepts, ideas or attacks efficient against lightweight ciphers may be applied to other, classical symmetric ciphers, or may be improved later.

In August 2018, NIST[1] started a competition whose goal is to standardize lightweight cryptographic algorithms which are suitable for use in situations where performance of the current NIST standards is not sufficient. There were 56 candidates submitted and selected to the first round of the competition in April 2019, of which 32 were announced to be selected to the second round in August 2019. Ten finalist candidates[2] were selected and announced in March 2021. The standardization process is not finished yet. It is expected that the results of the competition will be known later in 2022.

[1]National Institute of Standards and Technology

[2]Ciphers ASCON, Elephant, GIFT-COFB, Grain128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, TinyJambu, and Xoodyak were selected as the finalist candidates.

The cipher PRINCE[3] was published in 2012 in [1] by a team of creators from the Technical University of Denmark, INRIA[4], the Ruhr-University Bochum and NXP Semiconductors.

After the cipher had been published, the Ruhr-University Bochum announced a competition, called The PRINCE Challenge[5], to encourage finding practical attacks on round-reduced versions of PRINCE. Specifically, 4, 6, 8, 10 and 12 rounds of the cipher were considered in each category. There were two settings to choose from, either a chosen plaintext attack with at most 2^{20} chosen plaintexts or a known plaintext attack with at most 2^{30} known plaintexts. Several rounds of the competition were held until it ended in 2016.

In cases of four through six rounds of PRINCE, integral cryptanalysis seems to be the most powerful tool. After the challenge had started, integral distinguishers for four through six-round attacks were published in [2] and [3]. Integral attacks, along with several other techniques, were later improved in [4]. In cases of eight and more rounds, meet-in-the-middle attacks seem to be efficient. Arguably the most practical ones were published in [5].

Rounds	Time *	Data	Concept	Reference
4	$2^{7.5}$	2^6 CP	integral	[4], explained in Sec. 3.3
5	2^{13}	2^{13} CP	integral	[4], explained in Sec. 3.3
5	$2^{11.1}$	2^{11} CP	integral	Sec. 3.2.2, Sec. 3.3
6	$2^{26.5}$	2^{13} CP	integral	[4], explained in Sec. 3.4
6	$2^{24.6}$	2^{11} CP	integral	Sec. 3.2.2, Sec. 3.4
6	$2^{33.7}$	2^{16} CP	mitm	[5], explained in Sec. 4.1
7	$2^{44.3}$	2^{33} CP	h.-o. diff.	[4]
7	$2^{65.8}/2^{62.4}$	2^{16} CP	mitm	Section 4.3
8	$2^{82.4}/2^{49.7}$	2^{16} CP	mitm	[5], explained in Sec. 4.2

* Offline/Online complexity.

Table 1: A table of the current most practical attacks on round reduced versions of PRINCE. The time complexity is measured in encryption units.

The goal of this thesis is to survey several successful attacks from the above mentioned challenge and explain them in detail and, if possible, provide some contributions, such as new attacks, an extension of an existing attack or an explanation of nontrivial details omitted in the original papers. Specifically, the setting of chosen plaintext attack is considered. The thesis focuses on integral analysis and meet-in-the-middle attacks, which are the concepts that the current best attacks on round-reduced versions of the cipher use.

[3]A Low-latency Block Cipher for Pervasive Computing Applications

[4]The National Institute for Research in Digital Science and Technology

[5]Visit `https://informatik.rub.de/emsec/research/prince/` to see the homepage of the challenge.

1. Introduction to symmetric block ciphers

1.1 Construction of symmetric block ciphers

Let us first introduce the notation which we will use when speaking about ciphers.

A cipher consists of sets $\mathcal{P},\ \mathcal{C},\ \mathcal{K} \subseteq \{0,1\}^*$ and functions $\mathit{Enc}\colon \mathcal{P} \times \mathcal{K} \to \mathcal{C}$, $\mathit{Dec}\colon \mathcal{C} \times \mathcal{K} \to \mathcal{P}$ such that $\forall p \in \mathcal{P},\ \forall k \in \mathcal{K}\colon\ \mathit{Dec}\left(\mathit{Enc}\left(p,k\right),k\right) = p$.

The symbol $\mathcal{P}$ denotes the space of possible plaintexts - these are the messages that we might want to encrypt. The symbol $\mathcal{C}$ denotes the space of possible ciphertexts - these are the encrypted messages. The symbol $\mathcal{K}$ denotes the space of possible keys - these are the keys used for encrypting plaintexts and decrypting ciphertexts. The function *Enc* stands for encryption, the function *Dec* stands for decryption.

Ciphers can be categorized in several different ways. One possible categorization says how the encryption and decryption processes are related. We say that a cipher is symmetric if there is one key k which is shared among all involved parties, and is used for both encryption and decryption. Apart from symmetric ciphers, there are also asymmetric ciphers where the key comprises a public part used for encryption and a secret part used for decryption. This work, however, focuses only on symmetric ciphers.

If the set of possible plaintexts is restricted to plaintexts of certain fixed length ($\mathcal{P} = \{0,1\}^b,\ b \in \mathbb{N}$), we call the cipher a block cipher. The integer b denotes the block size of the cipher. The size of the key is often fixed as well, though it can be different from the block size.

Symmetric block ciphers are usually constructed by using one or several construction schemes. Some of the most commonly used ones are for example the substitution-permutation network (SPN) or the Feistel network[1].

In this section, we will focus on the construction schemes which are used in the PRINCE cipher. The details of how exactly are the construction schemes used will be presented later in Chapter 2.

1.1.1 Substitution-permutation network

A SPN consists of several substitution-permutation rounds. Each round comprises a key addition layer, a substitution layer and a permutation layer, except for the last round, which consists of a key addition layer, a substitution layer and another key addition layer.

[1]For example, DES (Data Encryption Standard) is one of the most famous ciphers which uses the Feistel network.

The key addition layer. In this layer, a *round key* (derived from the original key using a key schedule algorithm) is combined with the input, usually by performing the bitwise XOR operation (see Fig. 1.1).

$$\begin{tikzcd} & \text{round key} \arrow[d] & \\ \text{input} \arrow[r] & \bigoplus \arrow[r] & \text{output} \end{tikzcd}$$

Figure 1.1: A typical key addition layer

The substitution layer. This layer substitutes the input bits by different output bits. The substitution is usually performed by one or more substitution boxes (S-boxes). Each of them simply takes a block of bits and substitutes it by another block of bits of the same size. A typical substitution layer consists of several S-boxes of a small input size arranged in parallel. Figure 1.2 shows both the case with a single S-box and the case with several parallel S-boxes.

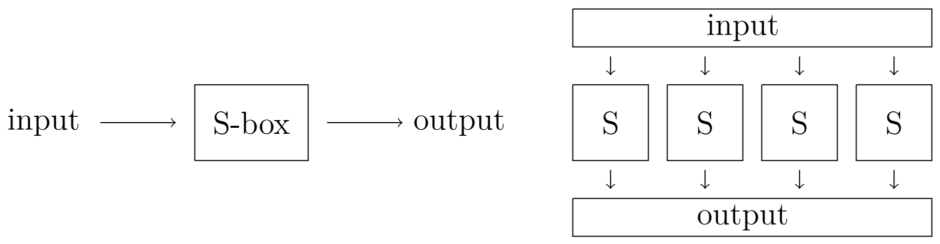


Figure 1.2: A scheme of a substitution layer

The permutation layer. The permutation layer simply permutes all input bits (see Fig. 1.3).

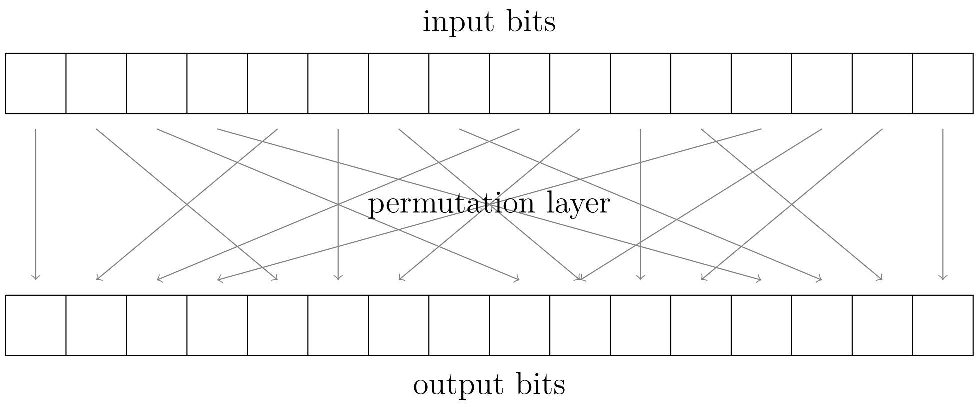


Figure 1.3: A scheme of a permutation layer

Altogether, several rounds of a SPN can achieve some of desired cryptographic properties, for example Shannon's confusion and diffusion properties[2]. We usually

[2]Shannon's confusion and diffusion properties were introduced in [6] in 1949.

want the S-box to have a property that changing one input bit results in a change of approximately half of the output bits. Every output bit of the S-box should also depend on every input bit. For the P-layer, we usually want the permutation to diffuse the output bits from one S-box to a lot of S-boxes of the next round.

A scheme of a 3-round SPN is depicted in Figure 1.4. As mentioned earlier, the last round of the network ends with a key addition layer instead of a permutation layer. This is because if there was no final key addition layer, the last S-layer and the last P-layer would be useless since both the substitution and the permutation are public. Moreover, having a P-layer in the last round of the network would be useless even with the final key addition layer present since we could simply permute the ciphertext and the final key.

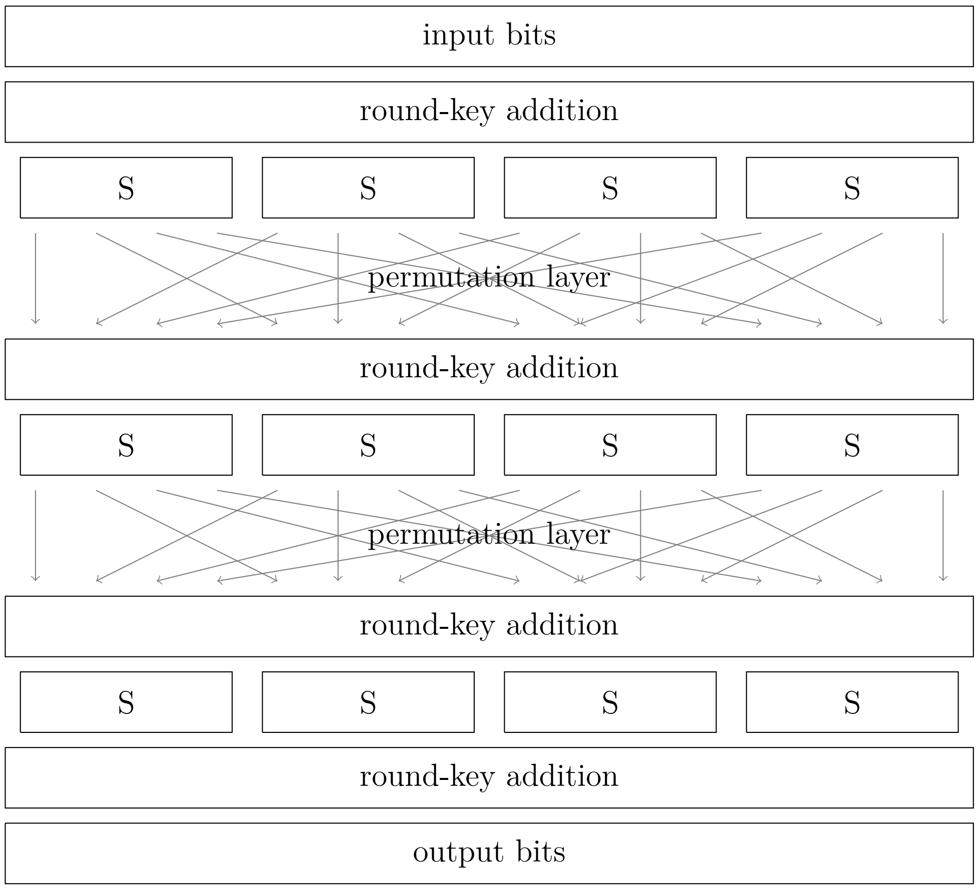


Figure 1.4: Multiple rounds of SPN

If we choose the S-boxes and the permutations carefully, the Shannon's confusion and diffusion properties will be satisfied after few rounds of the network. As we change one bit of the plaintext, several bits change during the first S-layer, the changes are distributed by the first P-layer into a lot of S-boxes of the second S-layer, after which most of the bits are already changed. The same effect occurs if we change one bit of the key.

1.1.2 FX construction

The FX construction is used to strengthen the original cipher, mostly to protect it from exhaustive key search attacks.

Apart from the original cipher (referenced to as the CORE cipher), two key addition layers are added. The plaintext is at first combined with a pre-whitening key (usually by bitwise XOR operation), then processed by the CORE cipher, and finally combined with a post-whitening key (again, usually by bitwise XOR operation). The pre-whitening and the post-whitening keys can be of a different size than the key of the CORE cipher. The scheme is depicted in Figure 1.5.

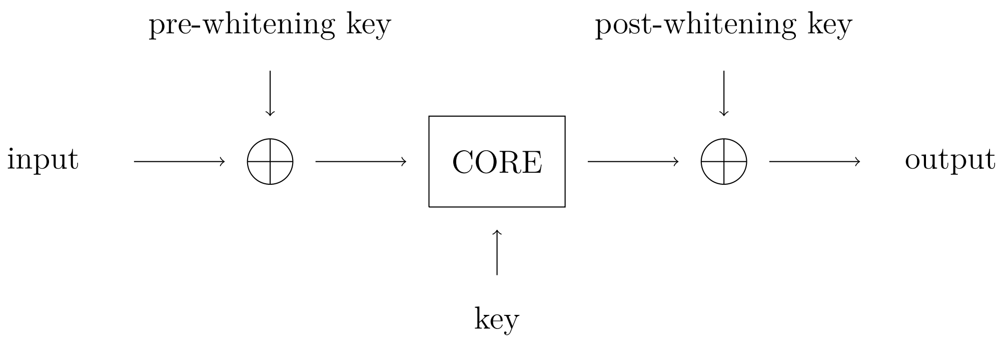


Figure 1.5: A scheme of the FX construction

1.2 Attacks on symmetric block ciphers

Symmetric block ciphers can be attacked in many different ways. This section provides a general description of the attacks and concepts studied later in the thesis.

Before we introduce the attacks themselves, let us first realize what goals do attacks have and what powers do attackers possess. A typical goal of an attack is to figure out the secret key, or at least a part of it. This can indeed be done by performing an exhaustive search - we can simply try all possible key values and see which one of them is correct. However, as the space of all possible keys gets bigger, this approach gets less practical, so we try to find different, more efficient methods of retrieving the key. The efficiency of the methods usually depends on our knowledge and our powers. The general taxonomy of attacks by attacker's powers is as follows.

If the attacker only knows some ciphertexts, we call this attack a *ciphertext only attack*. If the attacker knows multiple random plaintexts and their corresponding ciphertexts, we call the attack a *known plaintext attack*. If the attacker knows multiple plaintexts of his choice as well as the corresponding ciphertexts, we call the attack a *chosen plaintext attack*. If the attacker knows multiple ciphertexts of his choice along with the corresponding plaintexts, we call the attack a *chosen ciphertext attack*.

All the attacks studied later in this work fall into the category of chosen plaintext attacks.

When attacking symmetric block ciphers, we have a number of techniques and concepts at our disposal. Perhaps the best known techniques are the differential cryptanalysis, the linear cryptanalysis and the meet-in-the-middle attacks. The following sections introduce the concepts of meet-in-the-middle attacks and integral cryptanalysis which are later discussed in Chapters 3 and 4 in connection to PRINCE.

1.2.1 Integral cryptanalysis

The first concept we are going to introduce is the integral cryptanalysis. The integral cryptanalysis was presented in 2002 in [7] by Lars Knudsen[3]. The integral attack is a chosen plaintext attack where the general idea is to create a set of plaintexts with specific properties which we can track through several rounds of the cipher and see in which steps they hold. Typically, the set eventually loses all the properties as it goes through more rounds of the cipher, and we can use that to create an integral distinguisher.

Let us consider the following example. Suppose we have a cipher consisting of a four round substitution-permutation network with 64-bit plaintexts, ciphertexts and keys. Suppose there are sixteen 4-bit S-boxes arranged in parallel. Consider a set of sixteen plaintexts $\{0, \ldots, 15\}$. Note that the XOR sum of the whole set is zero. Suppose that the XOR sum of the whole set is also zero just before the last substitution layer of the cipher and the XOR sum of the whole set is generally unknown after the last substitution layer.

We can then retrieve the key easily as follows. First, we encrypt the sixteen plaintexts through the cipher. Second, we guess four bits of the key such that it enables us to decrypt the ciphertexts through the last key addition layer and through the last substitution layer. Third, we check if the XOR sum of the sixteen partially decrypted ciphertext is zero. If our guess is wrong, the XOR sum will not be equal to zero with high probability. On the other hand, if our guess is correct, the XOR sum has to be zero. Figure 1.6 depicts our situation.

Note the difference in efficiency in comparison to the exhaustive search. The exhaustive search requires trying 2^{64} values whereas the integral attack only requires $16 + 16 \times 2^4$ (approximately 2^8) encryptions or decryptions.

1.2.2 Meet-in-the-middle attacks

The meet-in-the-middle attack is a generic attack that enables us to trade the time complexity for the memory complexity. It was first introduced by W. Diffie and M. E. Hellman in 1997 in [9]. It is typically used against ciphers which consist of multiple subciphers that use different keys. The basic version of this attack can be performed as a known plaintext attack, however, having the powers of a chosen plaintext attack may lead to even more efficient versions of this attack.

[3]The work [7] gives a name to the concept and studies it in detail however the first basic integral attack was introduced a few years earlier in context of the cipher Square in [8].

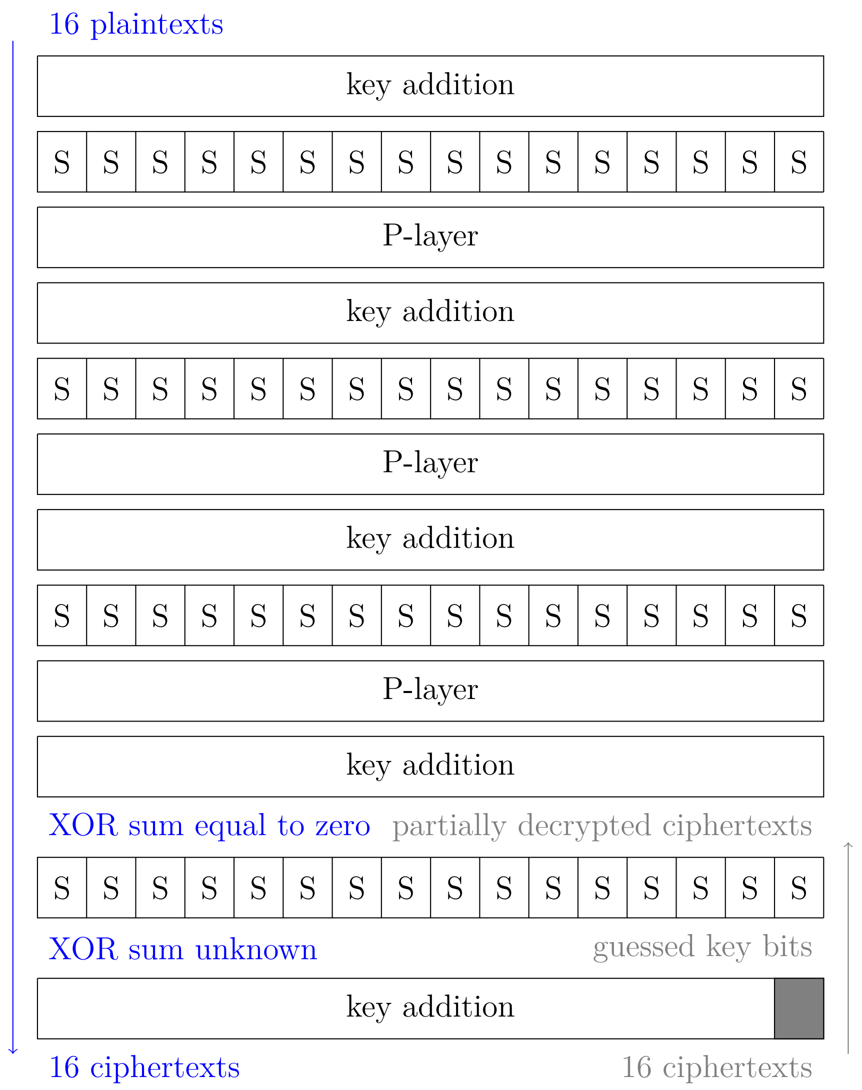


Figure 1.6: An example of an integral attack

As an example, let us consider a symmetric block cipher consisting of two subciphers using 64-bit keys k_a and k_b, respectively. If we have a single known plaintext, we can decrypt it through the first subcipher by all 2^{64} values of the key k_a and store the results in a lookup table. We can then ask for the full encryption of the plaintext and decrypt the ciphertext through the second subcipher by all 2^{64} values of the key k_b. If a decrypted value meets with a stored value from the lookup table, we can mark the corresponding values of k_a and k_b as key candidates. On the other hand, if a decrypted value cannot be found in the lookup table, the key k_b is incorrect. Figure 1.7 depicts the situation.

The memory complexity of such an attack is indeed 2^{64} stored values. The time complexity is 2×2^{64} encryptions or decryptions in contrast to the exhaustive search which would require searching through 2^{128} possible values of k_a and k_b.

The meet-in-the-middle attack can sometimes be divided into two phases - an offline phase and an online phase. The former represents a preprocessing phase which is only run once. The latter represents the actual attack which uses the results from the offline phase. The results from the offline phase can be used

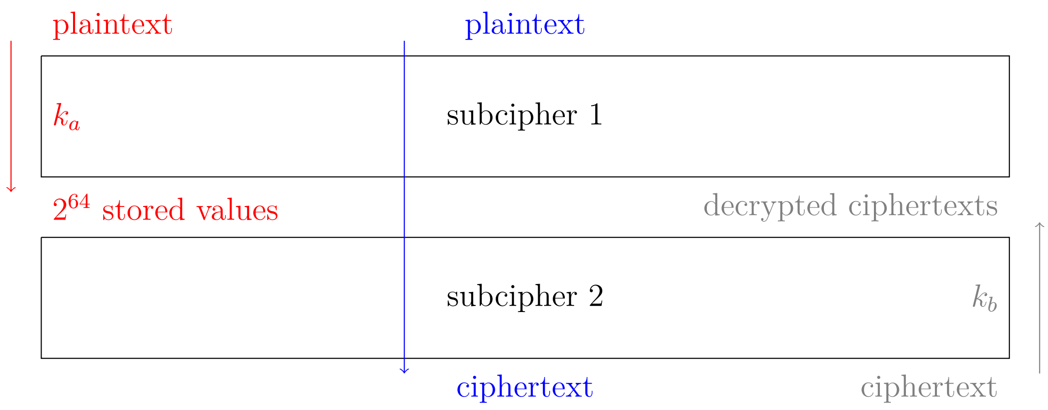


Figure 1.7: An example of a meet-in-the-middle attack

in several consecutive attacks, for example when recovering different secret keys.

2. PRINCE cipher description

PRINCE is a lightweight symmetric block cipher, first presented in [1] in 2012. In this chapter, we will describe the cipher and all its components in detail as it is presented in [1]. Our Python 3 reference implementation of the cipher is attached in A.1.

2.1 Overview

As we said earlier, PRINCE is a symmetric block cipher. The block size of PRINCE is 64 bits, the key size is 128 bits. In terms of the notation from Sec. 1.1, $\mathcal{P} = \mathbb{F}_2^{64}$, $\mathcal{C} = \mathbb{F}_2^{64}$, $\mathcal{K} = \mathbb{F}_2^{128}$.

The cipher is based on the FX construction. Three 64-bit sub-keys k_0, k_0' and k_1 are derived from the original key. The former two are used as the pre/post-whitening keys, the latter one is used as the key for the CORE cipher, as shown in Figure 2.1.

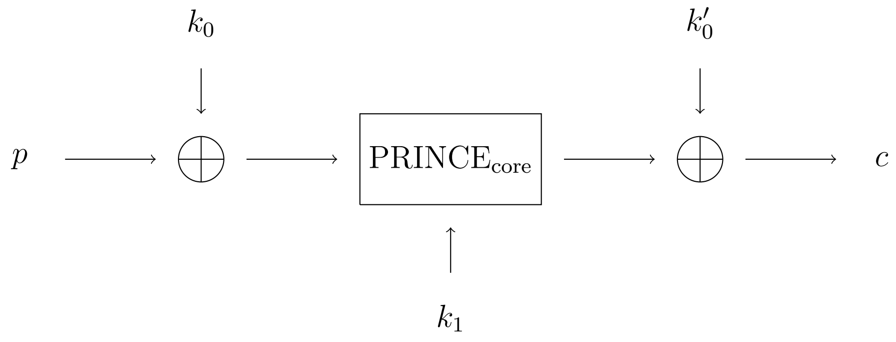

$$\begin{tikzcd} & k_0 \arrow[d] & & k_0' \arrow[d] & \\ p \arrow[r] & \oplus \arrow[r] & \boxed{\text{PRINCE}_{\text{core}}} \arrow[r] & \oplus \arrow[r] & c \\ & & k_1 \arrow[u] & & \end{tikzcd}$$

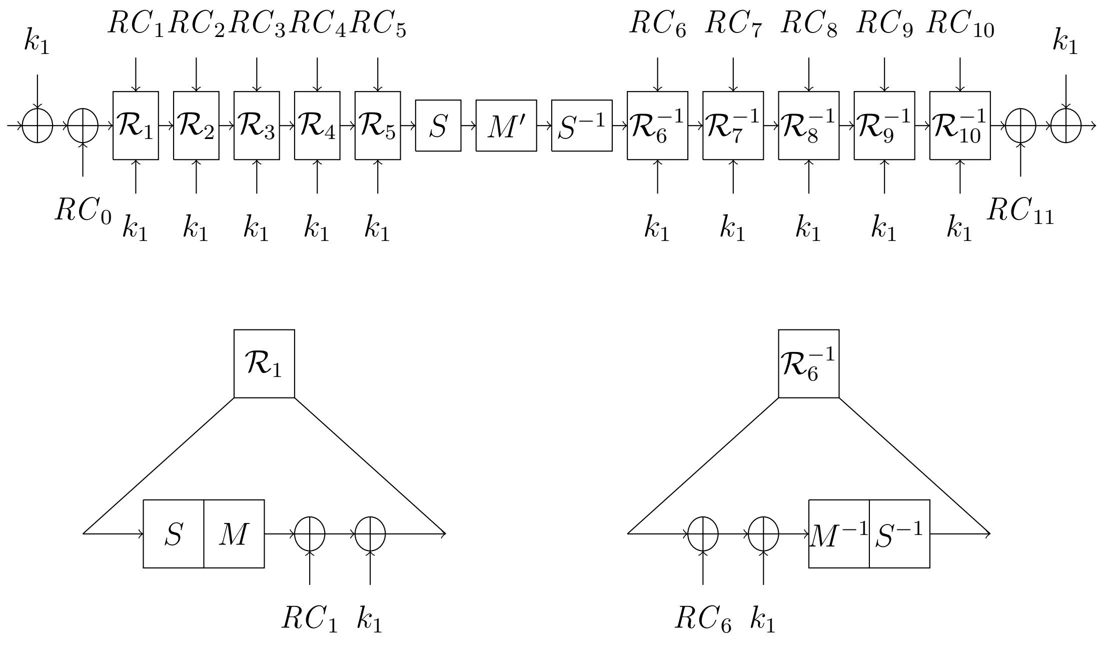
Figure 2.1: The scheme of PRINCE

2.1.1 The key schedule

PRINCE uses the following rules to derive the three sub-keys. At first, the original key k is split into two 64-bit halves, $k = k_0 \parallel k_1$. The third sub-key k_0' is then obtained as $k_0' = (k_0 \ggg 1) \oplus (k_0 \gg 63)$ where the $\ggg$ operator stands for the right circular shift and the $\gg$ operator stands for the right logical shift. We say that $(k_0 \parallel k_0' \parallel k_1)$ is the *extended key*.

2.2 PRINCE$_{\text{core}}$

The CORE cipher, referred to as PRINCE$_{\text{core}}$, is a 12-round SPN. Each round consists of a key addition layer, a round constant addition layer, a substitution layer and a linear layer. The PRINCE$_{\text{core}}$ key k_1 is used as a round key in every round of the SPN. The scheme of PRINCE$_{\text{core}}$ is depicted in Figure 2.2. The first five rounds will be occasionally referred to as the forward rounds, the next two rounds will be referred to as the middle rounds and the last five rounds will be referred to as the backward rounds.



k_1
RC_1 RC_2 RC_3 RC_4 RC_5
RC_6 RC_7 RC_8 RC_9 RC_{10}
k_1
$\mathcal{R}_1$ $\mathcal{R}_2$ $\mathcal{R}_3$ $\mathcal{R}_4$ $\mathcal{R}_5$
S M' S^{-1}
$\mathcal{R}_6^{-1}$ $\mathcal{R}_7^{-1}$ $\mathcal{R}_8^{-1}$ $\mathcal{R}_9^{-1}$ $\mathcal{R}_{10}^{-1}$
RC_0 k_1 k_1 k_1 k_1 k_1
k_1 k_1 k_1 k_1 k_1 RC_{11}
$\mathcal{R}_1$
S M
RC_1 k_1
$\mathcal{R}_6^{-1}$
M^{-1} S^{-1}
RC_6 k_1

Figure 2.2: The scheme of PRINCE$_{\text{core}}$

The key addition layer. The 64-bit state of the cipher is xored with the round key k_1.

The substitution layer. The PRINCE$_{\text{core}}$ uses the following 4-bit S-box. There are 16 such S-boxes arranged in parallel in every substitution layer.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	b	f	3	2	a	c	9	1	6	7	8	0	e	5	d	4

Figure 2.3: The PRINCE S-box

The linear layer. The 64-bit state of the cipher is multiplied by a 64×64 matrix M'. In most rounds, the sixteen 4-bit words of the cipher state are in addition permuted by a *shift rows* operation (meaning that $M = SR \circ M'$). The matrix M' and the sift rows operation SR will both be described later in Section 2.2.1.

The round constant addition layer. The 64-bit state of the cipher is xored with a 64-bit round constant RC_i (see Figure 2.4).

2.2.1 The linear layer

As we can see in Fig. 2.2, two different linear layers M and M' are used during the PRINCE encryption or decryption. The layer M' is only used once in

RC_0	0x0000000000000000
RC_1	0x13198a2e03707344
RC_2	0xa4093822299f31d0
RC_3	0x082efa98ec4e6c89
RC_4	0x452821e638d01377
RC_5	0xbe5466cf34e90c6c
RC_6	0x7ef84f78fd955cb1
RC_7	0x85840851f1ac43aa
RC_8	0xc882d32f25323c54
RC_9	0x64a51195e0e3610d
RC_{10}	0xd3b5a399ca0c2399
RC_{11}	0xc0ac29b7c97c50dd

Figure 2.4: The round constants of PRINCE

the middle of the encryption or decryption and it comprises a simple multiplication by the matrix M'. The layer M is used once in every round function $\mathcal{R}_i$ or $\mathcal{R}_i^{-1}$. It is defined as $M = SR \circ M'$, where SR is a permutation on the 16 four-bit words of the cipher state. The SR permutation is described in Figure 2.5.

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

$$\downarrow$$

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

Figure 2.5: The shift rows operation

To build the matrix M', we first define four 4×4 matrices M_0 through M_3.

$$M_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Next, we construct two 4×4 block matrices $\hat{M}^{(0)}$ and $\hat{M}^{(1)}$.

$$\hat{M}^{(0)} = \begin{pmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{pmatrix}, \quad \hat{M}^{(1)} = \begin{pmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{pmatrix}$$

Finally, we build a 64×64 block diagonal matrix using $(\hat{M}^{(0)}, \hat{M}^{(1)}, \hat{M}^{(1)}, \hat{M}^{(0)})$ as the diagonal blocks which we denote by M'.

Note that the matrix M' is an involution ($M'^{-1} = M'$) which allows the cipher to have the so called α-reflection property discussed in the following section.

2.2.2 The alpha-reflection property

PRINCE has an interesting property that allows using the same circuit for both encryption and decryption easily. We note that for all $0 \leq i \leq 11$ it holds that $RC_i \oplus RC_{11-i} = \text{0xc0ac29b7c97c50dd} =: \alpha$. Since M' is an involution and $RC_i \oplus RC_{11-i}$ is constant, the inverse of PRINCE$_{\text{core}}$ with a key k_1 is equal to PRINCE$_{\text{core}}$ with a key $(k_1 \oplus \alpha)$. We call this property *the α-reflection property.*

It follows that a PRINCE decryption with an extended key $(k_0 \| k_0' \| k_1)$ can be performed as a PRINCE encryption with an extended key $(k_0' \| k_0 \| k_1 \oplus \alpha)$.

2.3 Round reduced versions

When studying the strength of the cipher, it can be beneficial to study its reduced versions. There are several ways of how to reduce the number of rounds. Since the rounds of PRINCE are arranged symmetrically, the typical way (and perhaps the most natural one) is to cut the rounds out symmetrically. If we want to work with n-round reduced PRINCE, we only keep $\lceil (n-2)/2 \rceil$ forward rounds, two middle rounds and $\lfloor (n-2)/2 \rfloor$ backward rounds. Another possibility is to ignore the FX construction as well and focus on PRINCE$_{\text{core}}$ only. However, all attacks mentioned in Chapters 3 and 4 assume that the FX construction is preserved.

3. Integral attacks on PRINCE

The following chapters present some of the current most practical attacks on round reduced versions of PRINCE.

The first concept we are going to study is the integral cryptanalysis (see Section 1.2.1). The integral cryptanalysis is especially efficient against four, five and six rounds of PRINCE, the attacks in Sections 3.3 and 3.4 are the current most efficient chosen plaintext attacks on the corresponding versions of the cipher. The basic integral attacks were presented by P. Morawiecki in [2]. They were later improved by R. Posteuca and G. Negara in [3] and sped up by S. Rasoolzadeh and H. Raddum in [4]. This chapter provides the necessary theory, studies the 3.5-round distinguisher from [2], the 4.5-round distinguisher from [3], the faster key recovery technique from [4], and, based on the ideas from [3], it presents a new, even more efficient, 4.5-round integral distinguisher.

Apart from the new 4.5-round integral distinguisher, this chapter contributes to the research in the following ways. First, an inaccuracy in the original 3.5-round distinguisher is found and corrected. Second, more detailed description of all parts is given, including explicit algorithms for recovering the whole key of four, five and six round reduced versions of PRINCE. Third, a reference Python 3 implementation of some of the attacks is attached.

3.1 3.5-round integral distinguisher

This section studies the 3.5-round PRINCE integral distinguisher and the corresponding 4-round integral attack which were first introduced by P. Morawiecki in [2]. Specifically, this section presents the transformation of a PRINCE state into a square scheme, provides the elementary theory necessary for understanding the following attacks, explains the distinguisher in detail and gives a basic algorithm for a 4-round integral attack.

The square scheme. First, it is advantageous to look at the state of the cipher as a square scheme. Recall that PRINCE operates on 64-bit blocks. The 64-bit block can be divided into sixteen 4-bit parts that we can arrange in a 4×4 square scheme (see Figure 3.1). The 4-bit parts are also called nibbles and can be indexed by numbers 0 through 15.

For addressing individual nibbles or bits of the states, square brackets will be used. For example, if p is a PRINCE state, then $p[i]$ denotes the i-th nibble of the state and $p[j]_b$ denotes the j-th bit of the state.

Applying SPN layers. Applying the PRINCE layers can be done in the following way. The key addition and the round constant addition are straightforward, we can simply split the key (or the round constant) into sixteen words and XOR each of them with the corresponding nibble of the cipher state.

The substitution layer can be performed by applying the S-box to each nibble of the cipher state individually.

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

Figure 3.1: Arranging the PRINCE state into a square scheme

The linear layer is the most complicated one. Since the M' matrix is block diagonal with each block being a 16×16 matrix (for more details see Section 2.2.1), each 16-bit column of the square can be multiplied separately by one of the diagonal blocks $\hat{M}^{(0)}, \hat{M}^{(1)}$. To get the first or the fourth output column, we can apply the matrix $\hat{M}^{(0)}$ to the first (or the fourth) input column. To get the second or the third output column, we can apply the matrix $\hat{M}^{(1)}$ to the second (or the third) input column. The bitwise equations important for further analysis are as follows. An input column of nibbles (x_0, x_1, x_2, x_3) is transformed into an output column of nibbles (y_0, y_1, y_2, y_3), individual bits of the nibbles are denoted by superscripts.

$$
\begin{array}{llll}
\hat{M}^{(0)}: & & & \\
y_0^0 = x_1^0 \oplus x_2^0 \oplus x_3^0 & y_0^1 = x_0^1 \oplus x_2^1 \oplus x_3^1 & y_0^2 = x_0^2 \oplus x_1^2 \oplus x_3^2 & y_0^3 = x_0^3 \oplus x_1^3 \oplus x_2^3 \\
y_1^0 = x_0^0 \oplus x_1^0 \oplus x_2^0 & y_1^1 = x_1^1 \oplus x_2^1 \oplus x_3^1 & y_1^2 = x_0^2 \oplus x_2^2 \oplus x_3^2 & y_1^3 = x_0^3 \oplus x_1^3 \oplus x_3^3 \\
y_2^0 = x_0^0 \oplus x_1^0 \oplus x_3^0 & y_2^1 = x_0^1 \oplus x_1^1 \oplus x_2^1 & y_2^2 = x_1^2 \oplus x_2^2 \oplus x_3^2 & y_2^3 = x_0^3 \oplus x_2^3 \oplus x_3^3 \\
y_3^0 = x_0^0 \oplus x_2^0 \oplus x_3^0 & y_3^1 = x_0^1 \oplus x_1^1 \oplus x_3^1 & y_3^2 = x_0^2 \oplus x_1^2 \oplus x_2^2 & y_3^3 = x_1^3 \oplus x_2^3 \oplus x_3^3 \\
 & & & \\
\hat{M}^{(1)}: & & & \\
y_0^0 = x_0^0 \oplus x_1^0 \oplus x_2^0 & y_0^1 = x_1^1 \oplus x_2^1 \oplus x_3^1 & y_0^2 = x_0^2 \oplus x_2^2 \oplus x_3^2 & y_0^3 = x_0^3 \oplus x_1^3 \oplus x_3^3 \\
y_1^0 = x_0^0 \oplus x_1^0 \oplus x_3^0 & y_1^1 = x_0^1 \oplus x_1^1 \oplus x_2^1 & y_1^2 = x_1^2 \oplus x_2^2 \oplus x_3^2 & y_1^3 = x_0^3 \oplus x_2^3 \oplus x_3^3 \\
y_2^0 = x_0^0 \oplus x_2^0 \oplus x_3^0 & y_2^1 = x_0^1 \oplus x_1^1 \oplus x_3^1 & y_2^2 = x_0^2 \oplus x_1^2 \oplus x_2^2 & y_2^3 = x_1^3 \oplus x_2^3 \oplus x_3^3 \\
y_3^0 = x_1^0 \oplus x_2^0 \oplus x_3^0 & y_3^1 = x_0^1 \oplus x_2^1 \oplus x_3^1 & y_3^2 = x_0^2 \oplus x_1^2 \oplus x_3^2 & y_3^3 = x_0^3 \oplus x_1^3 \oplus x_2^3
\end{array}
\tag{3.1}
$$

Properties of nibbles. There are several properties that each nibble can have with respect to a set of PRINCE states, that we are interested in. We might not mention the set explicitly in further text if it is clear from the context or if we talk about a fixed set of PRINCE states. For the purpose of following definitions, let $k = 2^l$, $l \in \mathbb{Z}$, $l \geq 0$, let b denote the size of the nibbles in bits ($b = 4$ in case of PRINCE) and let $\Delta = \left\{x^{(0)}, \ldots, x^{(k-1)}\right\}$ denote the set of k PRINCE states (64-bit vectors).

Definition 1 (An active nibble)**.** *Let k, b, Δ be as described above. A nibble is called* active *(with respect to Δ), if it takes all 2^b possible values through the set Δ and each value is taken exactly $k/2^b$ times. We denote active nibbles by $\mathcal{A}$.*

Definition 2 (A quasi-active nibble)**.** *Let k, b, Δ be as described above. A nibble is called* quasi-active *(with respect to Δ), if it takes n distinct values ($n \in \mathbb{N}$) through the set Δ and each of the values is taken exactly k/n times. We denote quasi-active nibbles by $\mathcal{A}_n$.*

Remark. Notice that $\mathcal{A}$ denotes the same property as $\mathcal{A}_{2^b}$.

Definition 3 (A constant nibble)**.** *Let k, b, Δ be as described above. A nibble is called* constant *(with respect to Δ), if it only takes one distinct value through the set Δ. We denote constant nibbles by $\mathcal{C}$.*

Remark. Notice that $\mathcal{C}$ denotes the same property as $\mathcal{A}_1$.

Definition 4 (An even nibble)**.** *Let k, b, Δ be as described above. A nibble is called* even *(with respect to Δ), if each of the values taken through the set Δ occurs an even number of times. We denote even nibbles by $\mathcal{E}$.*

Remark. Similarly, we say that a nibble is odd with respect to a set of PRINCE states, if each of the values taken through the set occurs an odd number of times.

Definition 5 (A balanced nibble)**.** *Let k, b, Δ be as described above. A nibble i is called* balanced *(with respect to Δ), if the XOR sum of the values taken over the whole set Δ equals zero. That is, if $\bigoplus_{j=0}^{k-1} x^{(j)}[i] = 0$. We denote balanced nibbles by $\mathcal{B}$.*

Let us now take a look at the relations between the above mentioned properties. Recall that we consider $k = 2^l$, $l \in \mathbb{Z}$, $l \geq 0$, nibbles of bit size b, $\Delta = \left\{x^{(0)}, \ldots, x^{(k-1)}\right\}$ the set of k PRINCE states.

Observation 1. *An active nibble is balanced.*

Proof. It is sufficient to show that $\bigoplus_{i=0}^{2^b-1} i = 0$. Since the value of each bit (over the integers 0 through $2^b - 1$) is balanced, the equation holds.

$\square$

Observation 2. *An even nibble is balanced.*

Proof. Since $\forall z \in \mathbb{Z}_2^b \colon z \oplus z = 0$, occurrences of each value (which is taken even number of times) cancel out and the nibble is balanced.

$\square$

Corollary. A quasi-active nibble is also balanced.

Proof. A quasi-active nibble is by definition active or even. By Observations 1 and 2, it is also balanced.

$\square$

Remark. A constant nibble is balanced as well, since we only allow the set of PRINCE states to comprise an even number of states.

The 3.5-round distinguisher. We now have all necessary definitions and observations to describe the 3.5-round integral distinguisher from [2]. First, let us take a look at the parameters of the nibbles and the set of chosen plaintexts. As mentioned earlier, PRINCE nibbles are of bit size 4 (see Fig. 3.1). The set of plaintexts used for this attack is as follows. We (arbitrarily) choose one nibble to be active at the start and we let all other nibbles be constant. That is, we have a set of plaintexts where the value of one selected nibble varies through all possible values and the value of all other nibbles is constant. In terms of the preceding definitions, $k = 2^4 = 16$.

If we now track the set through four rounds of PRINCE, all nibbles are balanced in each step up to the last substitution layer, which destroys all of the properties. This observation can be used to form a 4-round attack. If we encrypt all 16 plaintexts of our set (by 4-round reduced PRINCE), we can then guess one nibble of the key $k_1 \oplus k_0'$ and partially decrypt the corresponding nibble of all 16 ciphertexts through the S-box of the fourth round. If we guess the key correctly, the decrypted values must be balanced. If our guess is incorrect, the decrypted values will not be balanced with high probability.

Before we formulate the algorithm precisely and discuss its complexity, let us first take a look at the properties of the nibbles with respect to our set in detail. Figure 3.2 shows the properties of the set of 16 chosen plaintexts through four rounds of PRINCE supposing the nibble 0 is chosen as the one active nibble at the start. Note that both key addition and round constant addition do not change any of the properties and are therefore omitted from the scheme. P. Morawiecki introduces the distinguisher in his work [2, Chap. 3]. However, the scheme provided in [2, Chap. 3, Fig. 2] is inaccurate. The following discussion shows that the nibbles of the third round are even but generally not quasi-active as the scheme in [2] states. This inaccuracy, however, does not affect the functionality of the distinguisher since the nibbles are still balanced after 3.5 rounds.

Theorem 3. *Consider a set Δ of 16 plaintexts with nibble* 0 *active and all other nibbles constant. Suppose we encrypt all plaintexts in Δ by four rounds of PRINCE. In each step of the encryption, the properties of nibbles (with respect to the set of the 16 corresponding states) are as depicted in Figure 3.2.*

Proof. As we have already mentioned, the key addition and the round constant addition layers preserve all properties. Since S-box is just a permutation, it preserves the $\mathcal{A}, \mathcal{A}_n, \mathcal{E}$ and $\mathcal{C}$ properties as well.

The only layer that is a bit more complicated is the linear layer. As mentioned earlier, the linear layer can be performed as applying matrices $\hat{M}^{(0)}$ and $\hat{M}^{(1)}$ to the columns of the square scheme individually.

In the first round, the constant columns remain constant and the column with the active nibble transforms into a column with all nibbles quasi-active. The former case is trivial. The latter case follows from a fact that the value of each output nibble depends only on 3 bits of the value of the active nibble. Thus,

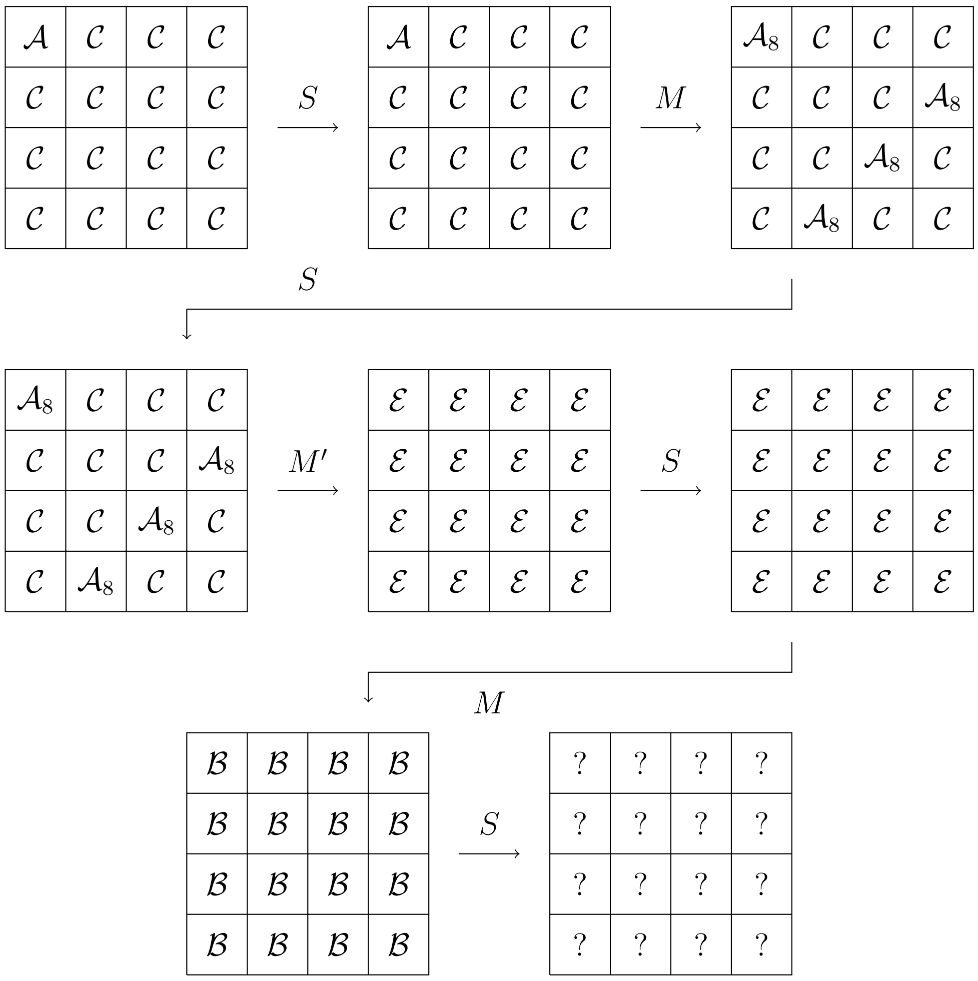


Figure 3.2: The scheme of the 3.5-round integral distinguisher

for each output nibble, two different input values of the active nibble produce the same output value.

In the second round, a column containing one quasi-active nibble and three constant nibbles is transformed into a column with all nibbles even. This is because the quasi-active nibble takes each distinct value exactly twice. For each output nibble, every such pair of equal input values is mapped to the same output value. Another pair of input values may be mapped to the same value as well however the number of occurrences of each output value in each output nibble stays even.

In the last round, a column with all nibbles even is transformed to a column with all nibbles balanced. Recall that each output bit of each output nibble is obtained as a XOR sum of three input bits. Since the input nibbles are even, if we iterate over the set of 16 plaintexts, each input bit is equal to one even number of times and is equal to zero even number of times. Therefore the XOR sum of each input bit taken over all 16 plaintexts equals zero and so does the XOR sum of each output bit.

The last S-box destroys the balance because no additional properties of the state with all nibbles balanced are guaranteed.

□

The basic 4-round attack. Algorithm 1 gives the basic 4-round integral attack based on the 3.5-round distinguisher described in Figure 3.2. The goal of the basic attack is to recover one nibble of the key $(k_1 \oplus k_0')$.

Algorithm 1 The basic 4-round integral attack

```
1: Let $\Delta = \{p^{(0)}, \dots, p^{(15)}\}$ be a set of plaintexts with one active nibble.
2: Encrypt each plaintext from $\Delta$ to get ciphertexts $C = \{c^{(0)}, \dots, c^{(15)}\}$.
3: for $i = 0, \dots, 15$ do
4:     for $j = 0, \dots, 15$ do
5:         Partially decrypt the attacked nibble $t$ of $c^{(j)}$ through the S-box of the
           last round using $i$ as the key $(k_1 \oplus k_0')[t]$ to get $m^{(j)}[t]$.
6:     end for
7:     if $\bigoplus_{j=0}^{15} m^{(j)}[t] = 0$ then
8:         Mark $i$ as a candidate for the key $(k_1 \oplus k_0')[t]$.
9:     end if
10: end for
```

Remark. Several candidates for the value of the attacked nibble of the key $(k_1 \oplus k_0')$ may remain after performing the attack once. In such case, we can repeat the attack with another set of plaintexts to eliminate the false key candidates.

Remark. If we want to recover the whole key $(k_1 \oplus k_0')$, we can simply perform Step 3 from Algorithm 1 sixteen times - once for each nibble.

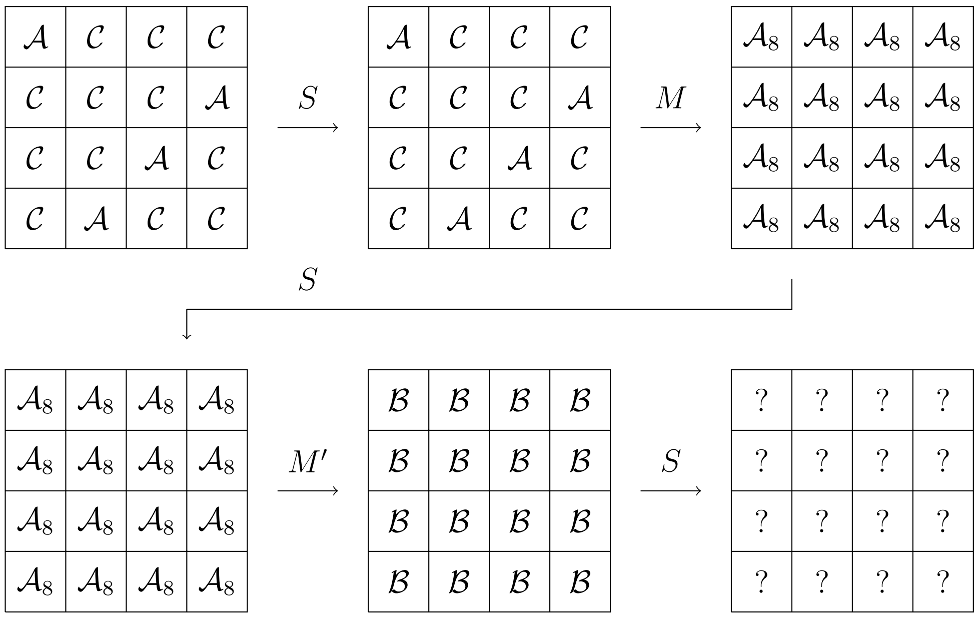
The complexity of this attack is as follows. We only need 16 chosen plaintexts. Each plaintext is encrypted once and then decrypted 16 times, which gives a time complexity of $2^4 + 2^8$ encryptions or decryptions.

The basic attack gives us an opportunity to recover the key $(k_1 \oplus k_0')$, however, it is not enough to figure out the original keys k_0 and k_1. To get the original keys as well, it suffices to find the round key k_1. The knowledge of the key $(k_1 \oplus k_0')$ empowers us to peel off the last round of the cipher so we would only need to attack a 3-round version of the cipher. This can be done using a 2.5-round integral distinguisher. The 2.5-round distinguisher requires a set of 16 chosen plaintexts with four nibbles active such that one inverse shift rows operation would transform the active positions to the same column. The scheme of the distinguisher is depicted in Figure 3.3 and its correctness follows from the correctness of the 3.5-round distinguisher (see Theorem 3).

Theorem 4. *Consider a set Δ of 16 plaintexts with nibbles $0, 7, 10, 13$ active and all other nibbles constant. Suppose we encrypt all plaintexts in Δ by three rounds of PRINCE. In each step of the encryption, the properties of nibbles (with respect to the set of the 16 corresponding states) are as depicted in Figure 3.3.*

Proof. The correctness of all steps involving a S-box follows directly from the fact that it preserves the $\mathcal{A}, \mathcal{A}_n, \mathcal{E}$ and $\mathcal{C}$ properties.

The correctness of the second step (the first linear layer) follows directly from Theorem 3. The only step left to study is the second linear layer. A column of



$\mathcal{A}$ $\mathcal{C}$ $\mathcal{C}$ $\mathcal{C}$ / $\mathcal{C}$ $\mathcal{C}$ $\mathcal{C}$ $\mathcal{A}$ / $\mathcal{C}$ $\mathcal{C}$ $\mathcal{A}$ $\mathcal{C}$ / $\mathcal{C}$ $\mathcal{A}$ $\mathcal{C}$ $\mathcal{C}$
S
$\mathcal{A}$ $\mathcal{C}$ $\mathcal{C}$ $\mathcal{C}$ / $\mathcal{C}$ $\mathcal{C}$ $\mathcal{C}$ $\mathcal{A}$ / $\mathcal{C}$ $\mathcal{C}$ $\mathcal{A}$ $\mathcal{C}$ / $\mathcal{C}$ $\mathcal{A}$ $\mathcal{C}$ $\mathcal{C}$
M
$\mathcal{A}_8$ (all 16 cells)
S
$\mathcal{A}_8$ (all 16 cells)
M'
$\mathcal{B}$ (all 16 cells)
S
? (all 16 cells)

Figure 3.3: The scheme of the 2.5-round integral distinguisher

quasi-active nibbles is transformed into a column of all nibbles balanced. However, since those quasi-active nibbles are even, the correctness of this step follows from the correctness of the corresponding step in Theorem 3.

The last S-box destroys the balance because no additional properties of the state with all nibbles balanced are guaranteed.

□

3.2 4.5-round integral distinguishers

This section studies two 4.5-round integral distinguishers. Both of them can be used for practical attacks on 5 and 6 rounds of PRINCE. The first one, studied in 3.2.1, was introduced by R. Posteuca and G. Negara in [3]. Although the work [3] introduces the distinguisher and the attacks, there is no scheme or an explanation of why the distinguisher holds provided. This section describes the distinguisher in detail, provides its basic scheme and discusses its functionality. The second distinguisher, presented in 3.2.2, is a new distinguisher, making the integral attacks on five and six rounds of PRINCE even faster.

3.2.1 The original 4.5-round integral distinguisher

This time, the set of chosen plaintexts is as follows. We choose three nibbles of the same column to take all possible 2^{12} values and we let all remaining nibbles be constant. Thus, we get a set of 2^{12} chosen plaintext which we then track through five rounds of PRINCE (see Fig. 3.4). Similarly to the 3.5-round distinguisher, all nibbles are balanced in each step up to the last substitution layer, which destroys all of the properties. This can be used to recover the key $k_1 \oplus k_0'$ nibble by nibble as in the 4-round attack.

Observation 5. *Consider a set Δ of 2^{12} plaintexts with nibbles $0, 1, 2$ taking all 2^{12} possible values and all other nibbles constant. Suppose we encrypt all plaintexts in Δ by five rounds of PRINCE. In each step of the encryption, the properties of nibbles (with respect to the set of the 2^{12} corresponding states) are as depicted in Figure 3.4.*

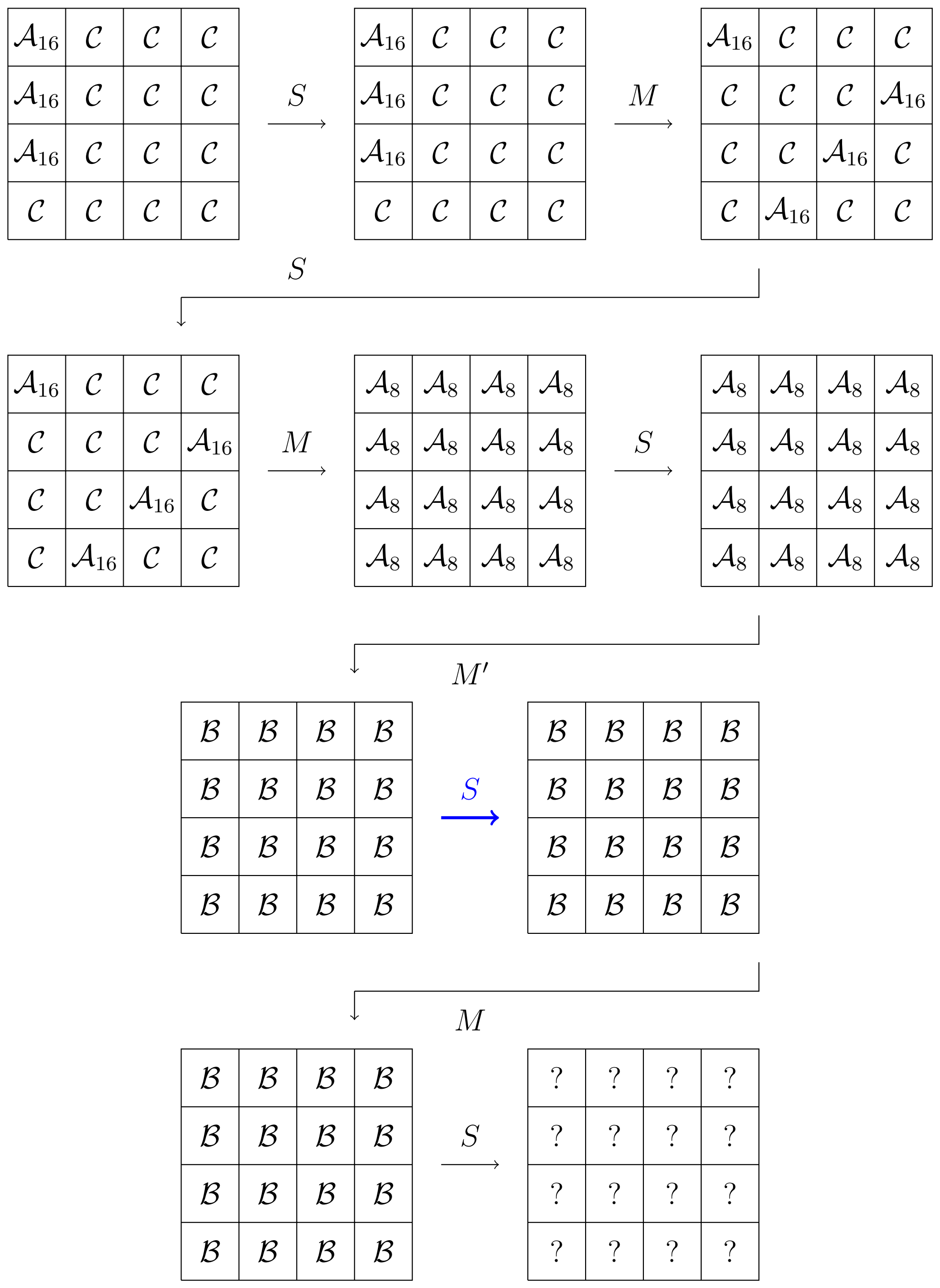


Figure 3.4: The scheme of the original 4.5-round integral distinguisher. The proof of the highlighted layer seems to be difficult.

There is no theoretical proof of why this should be true in the original paper. However, we can experimentally determine the following scheme of the distinguisher (Fig. 3.4) by running the corresponding attack many times and storing the intermediate states of the cipher. Based on our observations, we can verify the correctness of all steps of the distinguisher except for the fourth substitution layer.

Let us take a look at the scheme in detail. There are three active nibbles in the first row at the beginning. The first substitution layer does not change anything. The first matrix multiplication activates the fourth nibble of the first column as its value depends on all four nibbles of the column. The second substitution layer does not change anything, again. The second matrix multiplication activates all nibbles in each column. Each output nibble takes only eight distinct values because two input values of the one active nibble in each column produce the same output. Again, the third substitution layer does not change anything. The third and the fourth matrix multiplications keep all nibbles balanced (recall that quasi-active nibbles are always balanced, see Observation 2).

The only step we cannot verify by our observations is the fourth substitution layer. For the balance to be preserved through a substitution layer, it is not sufficient that the input nibbles are balanced, the input nibbles need to have some stronger property. Based on computer experiments, our hypothesis is that the extra property, allowing the balance to be preserved, is that the input nibbles are either even or odd (see Def. 4).

3.2.2 The new 4.5-round integral distinguisher

By computer experiments, we have found a new distinguisher which only requires a set of 2^{10} chosen plaintexts. The set of chosen plaintexts is as follows. We choose two nibbles of the same columns to take all possible 2^8 values. We set the two remaining nibbles of the column to be quasi-active, taking only two distinct values each. We let all remaining nibbles be constant. Thus, we get a set of 2^{10} chosen plaintexts which we can track through five rounds of PRINCE. As in the case of the previous distinguisher, all nibbles are balanced in each step up to the last substitution layer.

The scheme of such a distinguisher would be the same as in the case of the original 4.5-round distinguisher in Fig. 3.4, except for the first two states. The first two states are depicted in Figure 3.5. The correctness of all steps can be verified in the same way as it is discussed in the previous subsection. Again, our observations are not sufficient to verify that the fourth substitution layer preserves the balance.

$$\begin{array}{|c|c|c|c|}\hline \mathcal{A}_{16} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{16} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{2} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{2} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \end{array} \xrightarrow{S} \begin{array}{|c|c|c|c|}\hline \mathcal{A}_{16} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{16} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{2} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \mathcal{A}_{2} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \hline \end{array}$$

Figure 3.5: The first two states of the new 4.5-round integral distinguisher

The basic 5-round attack. Both the original and the new 4.5-round integral distinguishers can be used for the following 5-round attack. Algorithm 2 gives the basic 5-round integral attack based on the new 4.5-round distinguisher described in this subsection. The goal of the basic attack is to recover one nibble of the key $(k_1 \oplus k'_0)$.

Algorithm 2 The basic 5-round integral attack

```
 1: Let $\Delta = \{p^{(0)}, \dots, p^{(2^{10}-1)}\}$ be a set of plaintexts as described in Sec. 3.2.2.
 2: Encrypt each plaintext from $\Delta$ to get ciphertexts $C = \{c^{(0)}, \dots, c^{(2^{10}-1)}\}$.
 3: for $i = 0, \dots, 15$ do
 4:     for $j = 0, \dots, 2^{10} - 1$ do
 5:         Partially decrypt the attacked nibble $t$ of $c^{(j)}$ through the S-box of the
            last round using $i$ as the key $(k_1 \oplus k'_0)[t]$ to get $m^{(j)}[t]$.
 6:     end for
 7:     if $\bigoplus_{j=0}^{2^{10}-1} m^{(j)}[t] = 0$ then
 8:         Mark $i$ as a candidate for the key $(k_1 \oplus k'_0)[t]$.
 9:     end if
10: end for
```

Remark. Several candidates for the value of the attacked nibble of the key $(k_1 \oplus k'_0)$ may remain after performing the attack once. In such case, we can repeat the attack with another set of plaintexts to eliminate the false key candidates.

Remark. If we want to recover the whole key $(k_1 \oplus k'_0)$, we can simply perform Step 3 from Algorithm 2 sixteen times - once for each nibble.

The complexity of this attack is as follows. We need 2^{10} chosen plaintexts. Each plaintext is encrypted once and then decrypted 16 times, which gives a time complexity of $2^{10} + 2^{14}$ encryptions or decryptions.

The basic attack gives us an opportunity to recover the key $(k_1 \oplus k'_0)$, however, it is not enough to figure out the original keys k_0 and k_1. To get the original keys as well, it suffices to find the round key k_1. The knowledge of the key $(k_1 \oplus k'_0)$ empowers us to peel off the last round of the cipher so we would only need to attack a 4-round version of the cipher. This can be done by performing the 4-round attack from Section 3.1.

3.3 Faster key recovery technique

This section shows a technique of speeding up the attacks from previous sections and provides algorithms for full versions of four and five-round attacks. The faster key recovery method was introduced in and is taken from [4] by S. Rasoolzadeh and H. Raddum.

Recall that we have 16 nibbles of bit size b. This technique uses binary arrays of size 2^b. We only consider the case of PRINCE, so $b = 4$. For each nibble (for which we want to recover a part of the key), we define a 16-bit binary array A and we initially set all the bits to zero. As we encrypt messages from our set of chosen plaintexts, instead of guessing the key and decrypting the ciphertext through the last round of the cipher, we only switch one bit of A - the bit at the position corresponding to the ciphertext value. After we encrypt the whole set, we proceed to guessing the key and decrypting values through the last round, however, we only need to decrypt such values a, that $A[a] = 1$. Typically, not all 16 positions of A are set to 1 and thus we save some time.

The speed-up factor is studied in detail in [4]. It turns out that the average number of ciphertexts which need to be processed converges to approximately 6.5 for large sets of plaintexts. If we consider the 5-round attack from the previous section, this is a huge improvement since we would have to process all 2^{10} ciphertexts otherwise.

The full 4-round attack. We have already described the basic attack for recovering the key $(k_1 \oplus k_0')$ in Section 3.1. The full Algorithm 3 incorporates the faster key recovery technique from this section and shows how to recover keys k_1 and k_0 by peeling off the last round of the cipher and running the 2.5-round distinguisher as discussed in Section 3.1.

Algorithm 3 The full 4-round integral attack

```
1: Initialize sixteen 16-bit binary arrays $A_0, \dots, A_{15}$ (see Section 3.3).
2: Encrypt a set of $2^4$ plaintexts with one active nibble and flip the corresponding
   bits of $A_0, \dots, A_{15}$.
3: for each nibble $n$ do
4:     for all values $k$ of $(k_0' \oplus k_1)[n]$ do
5:         Partially decrypt all values $v$ where $A_n[v] = 1$ through the S-box of
   the last round using the guessed key $k$.
6:         Sum the partially decrypted values calculated in Step 5.
7:         If the sum equals zero, then $k$ remains a candidate for $(k_0' \oplus k_1)[n]$.
8:     end for
9: end for
10: Repeat Steps 1 through 9 until there is only one $(k_0' \oplus k_1)$ candidate for each
    nibble.
11:
12: Initialize sixteen 16-bit binary arrays $A_0, \dots, A_{15}$ (see Section 3.3).
13: Encrypt a set of $2^4$ plaintexts with four active nibbles (such that one inverse
    shift rows operation would shift the active positions to the same column),
    decrypt the ciphertexts through the last round of the cipher using $(k_0' \oplus k_1)$
    from Step 10 and flip the corresponding bits of $A_0, \dots, A_{15}$.
14: for each nibble $n$ do
15:     for all values $k$ of $k_1[n]$ do
16:         Partially decrypt all values $v$ where $A_n[v] = 1$ through the S-box of
    the third round using the guessed key $k$.
17:         Sum the partially decrypted values calculated in Step 16.
18:         If the sum equals zero, then $k$ remains a candidate for $k_1[n]$.
19:     end for
20: end for
21: Repeat Steps 12 through 20 until there is only one $k_1$ candidate for each
    nibble.
22:
23: Recover key $k_0$ from keys $k_1$ and $(k_1 \oplus k_0')$.
```

Let us investigate the complexity of the attack. Steps 2 and 13 require us to generate and encrypt 2^4 plaintexts. Thanks to arrays A, we only need to store 16×16 bits in memory. In Steps 5 and 16, we need to decrypt 6.5 ciphertexts on average (see Section 3.3) with 16 keys for all 16 nibbles. Steps 10 and 21 tell us to repeat the preceding steps until there is only one key candidate left for the whole key. It is shown in [4] by S. Rasoolzadeh and H. Raddum that it is sufficient to perform Steps 1-9 and 12-20 twice with very high probability.

Altogether, the data complexity is $2 \times 2 \times 2^4$ plaintexts, the memory complexity is saving the key candidates for each nibble plus saving arrays A. The time complexity is $2 \times 2 \times 2^4$ 4-round encryptions, 2×2^4 one-round decryptions and $2 \times 2 \times 16 \times 16 \times 6.5$ partial one-round decryptions, that is approximately $2^{7.5}$ 4-round PRINCE encryptions. To the best of our knowledge, this seems to be the fastest known attack on the 4-round reduced version of PRINCE.

The full 5-round attack. The five-round attack is similar to the four-round one, except it uses the 4.5-round distinguisher. Again, the basic version of Algorithm 4 has already been described in Section 3.2.2, faster key recovery technique from this section is incorporated and recovering k_1 and k_0 by peeling off the last round and running the 4-round attack is included as well.

Similarly to the 4-round case, the data complexity of the 5-round attack is $2 \times 2^{10} + 2 \times 2^4$ plaintexts, the memory complexity is saving the key candidates for each nibble plus saving arrays A. The time complexity is $2 \times 2^{10} + 2 \times 2^4$ 5-round encryptions, 2×2^4 one-round decryptions and $2 \times 2 \times 16 \times 16 \times 6.5$ partial one-round decryptions, that is approximately $2^{11.1}$ 5-round PRINCE encryptions. To the best of our knowledge, this seems to be the fastest known attack on the 5-round reduced version of PRINCE.

Remark. In Step 13, we can actually use a subset of the set of plaintexts used in Step 2 instead of generating new 2^4 plaintexts. However, it requires saving the corresponding ciphertexts of the subset in Step 2.

Python 3 implementations of both basic and full four and five-round attacks and their variants are provided as an attachment in A.2.

3.4 6-round integral attack

After explaining the four and five-round attacks, we conclude this chapter by investigating how an extra round of the cipher can be attacked and discussing the six-round attack algorithm.

The full 6-round attack. The six-round attack described by Algorithm 5 is similar to the five-round one, except it extends the attack for one additional round. To be able to partially decrypt one more round of the cipher (so we could use the 4.5-round distinguisher), we need to guess four nibbles of the last round key. If we do so, we can decrypt a column of the ciphertexts through the extra round of the cipher and use the 4.5-round distinguisher (along with the faster key recovery technique) to recover the round key k_1.

Algorithm 4 The full 5-round integral attack

```
1: Initialize sixteen 16-bit binary arrays $A_0, \dots, A_{15}$ (see Section 3.3).
2: Encrypt a set of $2^{10}$ plaintexts as described in Sec. 3.2.2 and flip the corresponding bits of $A_0, \dots, A_{15}$.
3: for each nibble $n$ do
4:     for all values $k$ of $(k'_0 \oplus k_1)[n]$ do
5:         Partially decrypt all values $v$ where $A_n[v] = 1$ through the S-box of the last round using the guessed key $k$.
6:         Sum the partially decrypted values calculated in Step 5.
7:         If the sum equals zero, then $k$ remains a candidate for $(k'_0 \oplus k_1)[n]$.
8:     end for
9: end for
10: Repeat Steps 1 through 9 until there is only one $(k'_0 \oplus k_1)$ candidate for each nibble.
11:
12: Initialize sixteen 16-bit binary arrays $A_0, \dots, A_{15}$ (see Section 3.3).
13: Encrypt a set of $2^4$ plaintexts with one active nibble, decrypt the ciphertexts through the last round of the cipher using $(k'_0 \oplus k_1)$ from Step 10 and flip the corresponding bits of $A_0, \dots, A_{15}$.
14: for each nibble $n$ do
15:     for all values $k$ of $k_1[n]$ do
16:         Partially decrypt all values $v$ where $A_n[v] = 1$ through the S-box of the fourth round using the guessed key $k$.
17:         Sum the partially decrypted values calculated in Step 16.
18:         If the sum equals zero, then $k$ remains a candidate for $k_1[n]$.
19:     end for
20: end for
21: Repeat Steps 12 through 20 until there is only one $k_1$ candidate for each nibble.
22:
23: Recover key $k_0$ from keys $k_1$ and $(k_1 \oplus k'_0)$.
```

Remark. Note that in Step 10, if there is a nibble with no key candidates, the key K cannot be the correct column key and we can therefore skip to the next value of K immediately.

Similarly to the previous attacks, the data complexity is 2×2^{10} plaintexts, the memory complexity is saving 2^{10} ciphertexts, saving key candidates and saving arrays A. The time complexity is 2×2^{10} 6-round encryptions, $2 \times 4 \times 2^{16} \times 2^{10}$ partial (one-column) one-round decryptions and $2 \times 4 \times 2^{16} \times 4 \times 2^4 \times 6.5$ partial (one-nibble) one-round decryptions, that is approximately $2^{24.6}$ 6-round PRINCE encryptions. To the best of our knowledge, this seems to be the fastest known attack on the 6-round reduced version of PRINCE.

Algorithm 5 The full 6-round integral attack

```
1: Initialize sixteen 16-bit binary arrays $A_0, \dots, A_{15}$ (see Section 3.3).
2: Encrypt a set of $2^{10}$ plaintexts as described in Sec. 3.2.2.
3: for each column $c$ do
4:     for all values $K$ of $(k_0' \oplus k_1)$ column $c$ do
5:         Decrypt the column $c$ of the ciphertexts from Step 2 through the last round using the key $K$ and flip the corresponding bits of the corresponding arrays $A$.
6:         for each corresponding nibble $n$ do
7:             for all values $k$ of $k_1[n]$ do
8:                 Partially decrypt all values $v$ where $A_n[v] = 1$ through the S-box of the fifth round using the guessed key $k$.
9:                 Sum the partially decrypted values calculated in Step 8.
10:                If the sum equals zero, then $k$ remains a candidate for $k_1[n]$ and $K$ remains a candidate for $(k_0' \oplus k_1)$ column $c$.
11:            end for
12:        end for
13:    end for
14: end for
15: Repeat Steps 1 through 14 until there is only one $(k_0' \oplus k_1)$ and $k_1$ candidate for each nibble.
16:
17: Recover key $k_0$ from keys $k_1$ and $(k_1 \oplus k_0')$.
```

4. Meet-in-the-middle attacks on PRINCE

This chapter studies meet-in-the-middle attacks (see Section 1.2.2) on PRINCE in detail. The six and eight-round attacks have been introduced by P. Derbez and L. Perrin in [5]. The chapter starts with a 6-round meet-in-the-middle attack and moves on to an 8-round meet-in-the-middle attack, which is arguably the current best attack on 8 round reduced PRINCE. An extension of these attack to a 7-round attacks is mentioned and the chapter is concluded with a new 7-round meet-in-the-middle attack.

The theory of Sections 4.1 and 4.2, the corresponding attacks and the graphics of the corresponding figures is taken from the original paper [5]. This chapter contributes to the research in the following ways. First, the proofs of the meet-in-the-middle criteria of the six and eight-round attacks are explained in better detail. Second, an explicit, more detailed description of how to recover the full key in case of six rounds is provided in Attachment A.3. Third, attacks on 7 rounds of PRINCE are discussed. An extension of the previous attacks is presented and a new 7-round meet-in-the-middle attack with low data complexity is proposed.

4.1 6-round meet-in-the-middle attack

This section studies the 6-round meet-in-the-middle attack from [5]. Although the integral attack from Section 3.3 is more practical, understanding this attack helps with understanding all remaining meet-in-the-attacks from the following sections.

It is once again convenient to look at the PRINCE states as a square scheme, as discussed in Section 3.1 in Figure 3.1. Moreover, we will denote states of the cipher by symbols x_i, x'_i, y_i and y'_i in such way that x_i is the state before the i-th S-box, y_i is the state after the i-th S-box, y'_i is the state before the $(n+1-i)$-th S-box and x'_i is the state after the $(n+1-i)$-th S-box, where n is the total number of S-boxes. The notation is illustrated in Figure 4.1. Recall that square brackets are used for addressing individual nibbles or bits of the states.

Let us start with definitions of specific sets of PRINCE states that will be used among these meet-in-the-middle attacks.

Definition 6 (A δ-set)**.** *Consider* Δ *a set of* 16 *PRINCE states. We call such set* a δ-set*, if there are exactly one active nibble and* 15 *constant nibbles with respect to* Δ*.*

Remark. Recall that a PRINCE state a is an element of $\{0,1\}^{64}$. In the notation of nibbles, the state a can be written as $a = a[0] \parallel \ldots \parallel a[15]$, where $\forall i \in \{0,...,15\}\colon a[i] \in \{0,1\}^4$. A set Δ of 16 states $\{a^{(0)}, \ldots, a^{(15)}\}$ is a δ-set if $\exists!\ j \in \{0,\ldots,15\}\colon \forall i \in \{0,...,15\},\ i \neq j\colon a^{(0)}[i] = a^{(1)}[i] = \cdots = a^{(15)}[i]$ and $\forall\ k,l \in \{0,\ldots,15\},\ k \neq l\colon a^{(k)}[j] \neq a^{(l)}[j]$.

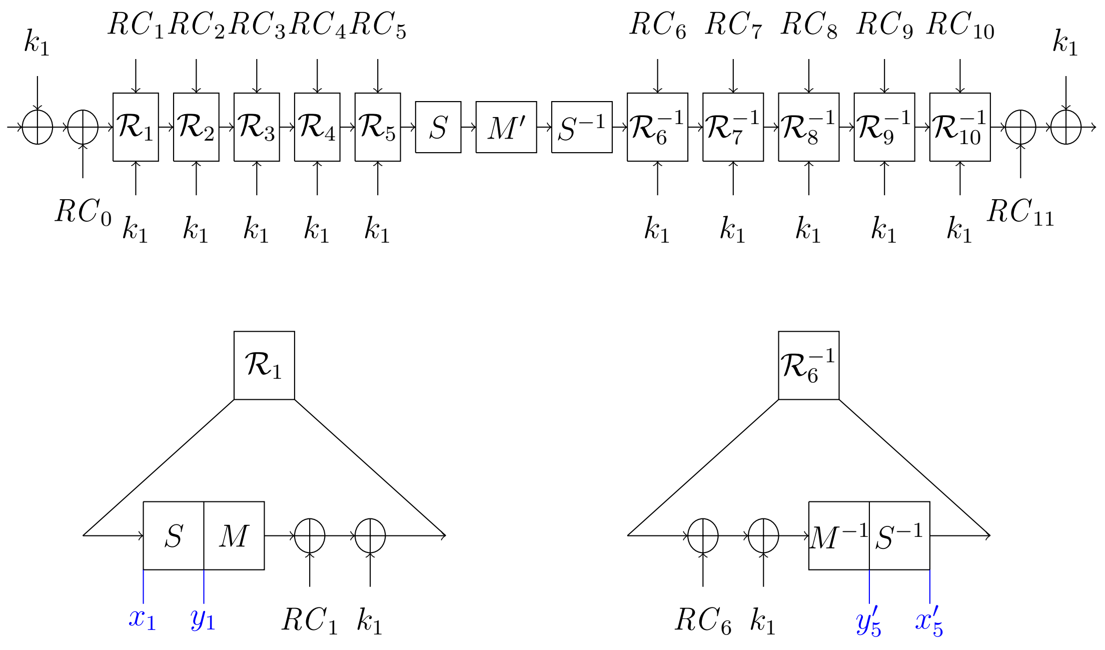

k_1
$RC_1 RC_2 RC_3 RC_4 RC_5$
RC_6 RC_7 RC_8 RC_9 RC_{10}
k_1
$\mathcal{R}_1$ $\mathcal{R}_2$ $\mathcal{R}_3$ $\mathcal{R}_4$ $\mathcal{R}_5$ S M' S^{-1} $\mathcal{R}_6^{-1}$ $\mathcal{R}_7^{-1}$ $\mathcal{R}_8^{-1}$ $\mathcal{R}_9^{-1}$ $\mathcal{R}_{10}^{-1}$
RC_0 k_1 k_1 k_1 k_1 k_1
k_1 k_1 k_1 k_1 k_1 RC_{11}
$\mathcal{R}_1$
S M
x_1 y_1 RC_1 k_1
$\mathcal{R}_6^{-1}$
M^{-1} S^{-1}
RC_6 k_1 y'_5 x'_5

Figure 4.1: The notation of the PRINCE states

Definition 7 (A structured δ-set)**.** *Let* $\Delta = \{a^{(0)}, \dots, a^{(15)}\}$ *be a* δ*-set where the nibble* j *is the one active nibble. We call* Δ a structured δ-set, *if*

$$\forall i = 1, \dots, 15\colon a^{(0)}[j] \oplus a^{(i)}[j] = i.$$

The following attack considers a δ-set with the nibble 8 chosen as the active nibble. The attack relies on the two following observations. Figure 4.2 depicts the situation from Observation 6.

Observation 6. *Consider a set of* 16 *plaintexts* $\{p^{(0)}, \dots, p^{(15)}\}$ *and its encryption through 6-round reduced PRINCE. If the set* $\{y_2^{(0)}, \dots, y_2^{(15)}\}$ *is a structured* δ*-set, then the sequence* $[y_2'^{(1)}[8] \oplus y_2'^{(0)}[8], \dots, y_2'^{(15)}[8] \oplus y_2'^{(0)}[8]]$ *is fully determined by nibbles* $x_3^{(0)}[2,5,8,15]$ *and* $x_3'^{(0)}[2,5,8,15]$*. In other words, there are only* 2^{32} *possible values of such 60-bit sequence.*

Proof. Since we know the differences

$$y_2^{(0)} \oplus y_2^{(i)} \ \forall i = 0, \dots, 15,$$

we can compute the values of

$$x_3^{(i)}[2,5,8,15] \oplus x_3^{(0)}[2,5,8,15] \ \forall i = 1, \dots, 15$$

through one M-layer and one key addition layer.

Choosing the value of $x_3^{(0)}[2,5,8,15]$ allows us to compute the values of

$$x_3^{(i)}[2,5,8,15] \ \forall i = 0, \dots, 15.$$

Since $y_3 = S(x_3)$, we also know the values of

$$y_3^{(i)}[2,5,8,15] \ \forall i = 0, \dots, 15.$$

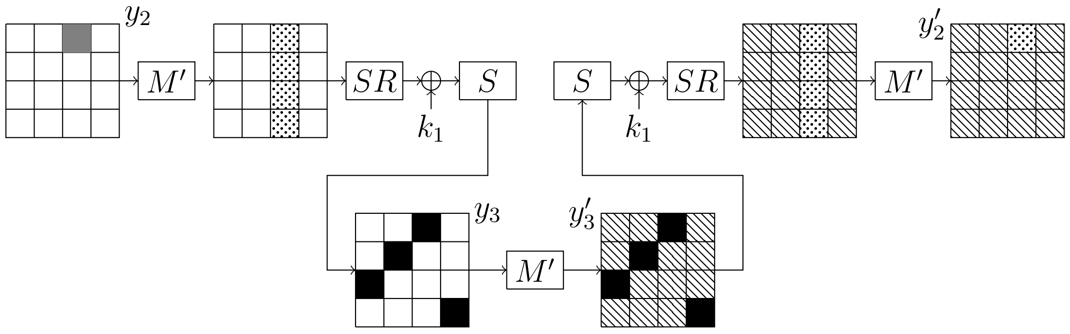


Figure 4.2: The scheme of Observation 6. White nibbles are constant. Gray nibble is active. Black nibbles serve as parameters. The difference of dotted nibbles is known. Hatched nibbles play no role.

The middle M'-layer gives us the values of

$$y_3'^{(i)}[2,5,8,15] \oplus y_3'^{(0)}[2,5,8,15] \ \forall i = 1,\dots,15,$$

and by choosing the values of $x_3'^{(0)}[2,5,8,15]$, we can compute $y_3'^{(0)}[2,5,8,15]$, and consequently

$$y_3'^{(i)}[2,5,8,15] \ \forall i = 0,\dots,15, \ \ x_3'^{(i)}[2,5,8,15] \ \forall i = 0,\dots,15.$$

Thus we can compute the values of $y_2'^{(i)}[8] \oplus y_2'^{(0)}[8] \ \forall i = 1,\dots,15$ through one key addition layer and one M-layer.

□

Observation 7. *The values of nibbles $(k_0 \oplus k_1)[8..11], k_1[8]$ and $(k_0' \oplus k_1)[8..11]$ of the PRINCE keys can be computed from the following* 33 *key bits:*

$$(k_0 \oplus k_1)[36..47]_b, (k_0' \oplus k_1)[36..47]_b, k_1[32..35]_b, k_0[31..35]_b.$$

Proof. Thanks to the key schedule algorithm of PRINCE (see Sec. 2.1.1), the value of $k_0'[32..35]_b$ can be computed from $k_0[31..34]_b$. The values of $(k_0 \oplus k_1)[32..35]_b$ and $(k_0' \oplus k_1)[32..35]_b$ can be computed from $k_0'[32..35]_b$, $k_0[32..35]_b$ and $k_1[32..35]_b$.

□

The 6-round attack. We now have all necessary observations for forming a 6-round chosen plaintext attack. The attack can be divided into an offline phase and an online phase. In the offline phase, we compute and store all 2^{32} 60-bit sequences from Observation 6. The online phase goes as follows. We generate 2^{16} chosen plaintexts such that nibbles 8 through 11 take all possible values and all other nibbles are constant and we encrypt all of the plaintexts through 6-round reduced PRINCE to get 2^{16} ciphertexts. We arbitrarily choose one of the plaintexts to be $p^{(0)}$. We guess several bits of $(k_0 \oplus k_1)$ and k_1 to obtain a nibble of the state $y_2^{(0)}$. We create a structured δ-set $\{y_2^{(0)},\dots,y_2^{(15)}\}$ and decrypt the set back to get a set of plaintexts $\{p^{(0)},\dots,p^{(15)}\}$ (which will be a subset of the original 2^{16} chosen plaintexts). We find the corresponding ciphertexts $\{c^{(0)}, \ldots, c^{(15)}\}$. We guess several bits of $(k_0' \oplus k_1)$ and k_1 to obtain a nibble of the states $y_2'^{(i)}$ $\forall i = 0, \ldots, 15$ and compute the sequence $[y_2'^{(0)}[8] \oplus y_2'^{(1)}[8], \ldots, y_2'^{(0)}[8] \oplus y_2'^{(15)}[8]]$. If there is such sequence among the 2^{32} possible sequences stored from the offline phase, then the guessed key bits are candidates for the actual key bits. The attack is depicted in Figure 4.3 and Algorithm 6 gives its pseudocode.

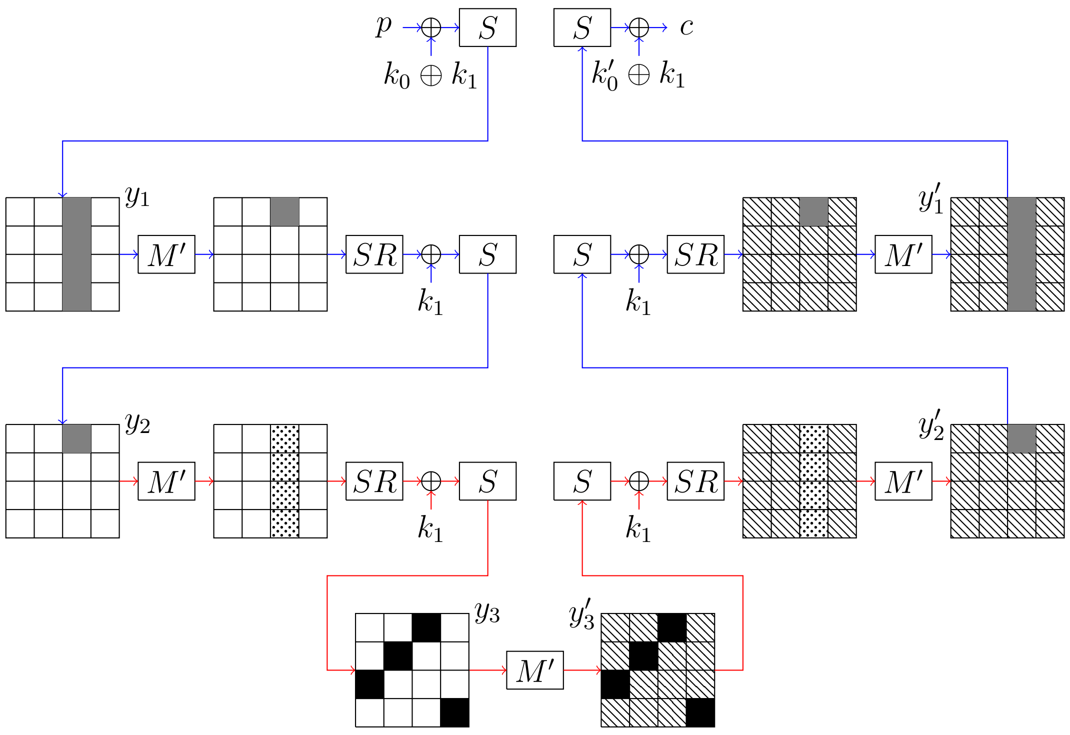


Figure 4.3: The 6-round meet-in-the-middle attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

Let us inspect the complexity of the attack. First, the probability of an incorrect guess to pass the meet-in-the-middle criterion is $2^{32} \times 2^{-60} = 2^{-28}$. Since we guess 2^{33} key bits, we expect to get 2^5 key candidates after one iteration of the attack and only one key candidate after two iterations of the attack.

The data complexity is 2^{16} chosen plaintexts. The memory requirements are $2^{32} \times 15 \times 4$ bits to store the sequences from the offline phase and $2^{16} \times 64$ bits to store the ciphertexts during the online phase. The time complexity of the online phase is 2^{16} 6-round PRINCE encryptions for producing ciphertexts plus $(2^{33}+2^5)\times 16$ encryptions through 10 S-boxes for creating the δ-sets and the states y_2'. Altogether, it is approximately $2^{33.7}$ 6-round PRINCE encryptions. The time complexity of the offline phase is computing the 2^{32} sequences, which requires encryption through $2^{16} \times 16 \times 4 + 2^{32} \times 16 \times 4$ S-boxes, which is approximately $2^{31.4}$ 6-round PRINCE encryptions.

Note that this attack only recovers 33 out of 128 key bits. To recover the remaining 95 key bits, we might run different variants of the attack without signif-

Algorithm 6 The 6-round meet-in-the-middle attack

```
1: Precompute the set $S$ of all $2^{32}$ possible sequences of
   $$[y_2'^{(0)}[8] \oplus y_2'^{(1)}[8], \dots, y_2'^{(0)}[8] \oplus y_2'^{(15)}[8]].$$
2: Set $P$ a set of $2^{16}$ chosen plaintexts such that nibbles $8-11$ take all possible
   values and all other nibbles are constant.
3: Encrypt all plaintexts in $P$ to get $2^{16}$ ciphertexts.
4: Arbitrarily choose a plaintext $p^{(0)} \in P$.
5: for all $2^{33}$ values of $(k_0 \oplus k_1)[8..11]$, $(k_0' \oplus k_1)[8..11]$ and $k_1[8]$ do
6:     Find $y_2^{(0)}[8]$.
7:     For $i = 1, \dots, 15$ set $y_2^{(i)}[8]$ so that $y_2^{(i)}[8] \oplus y_2^{(0)}[8] = i$.
8:     Find $p^{(i)}$ $\forall i = 1, \dots, 15$.
9:     Find the corresponding ciphertexts $c^{(i)}$ $\forall i = 0, \dots, 15$.
10:    Find $y_2'^{(i)}[8]$ $\forall i = 0, \dots, 15$.
11:    Compute the sequence $s = [y_2'^{(0)}[8] \oplus y_2'^{(1)}[8], \dots, y_2'^{(0)}[8] \oplus y_2'^{(15)}[8]]$.
12:    If $s \notin S$, then the guessed value is not a key candidate.
13: end for
14: Repeat Steps 4 through 13 until there is only one key candidate.
```

icantly increasing the overall time and data complexity. The idea behind the attacks remains the same as the idea behind the attack studied in this section, only different nibbles are considered. Example schemes of attacks leading to recovery of the whole key are provided in Attachment A.3.

4.2 8-round meet-in-the-middle attack

This section studies the 8-round meet-in-the-middle attack from [5]. The 8-round attack is similar to the previous one. Again, it considers a δ-set with the nibble 8 chosen as the active nibble and relies on the following observations. Figure 4.4 depicts the situation from Observation 8.

Observation 8. *Consider a set of* 16 *plaintexts* $\{p^{(0)}, \dots, p^{(15)}\}$ *and its encryption through 8-round reduced PRINCE. If the set* $\{M(y_1^{(0)}), \dots, M(y_1^{(15)})\}$ *is a structured* δ*-set, then the sequence*

$$[M(y_1'^{(1)})[8] \oplus M(y_1'^{(0)})[8], \dots, M(y_1'^{(15)})[8] \oplus M(y_1'^{(0)})[8],$$
$$y_2'^{(1)}[9] \oplus y_2'^{(0)}[9], \dots, y_2'^{(15)}[9] \oplus y_2'^{(0)}[9]]$$

is fully determined by nibbles:

$$x_3^{(0)}[2,5,8,15],\ x_3'^{(0)}[2,5,8,15],\ x_2^{(0)}[8],\ x_2'^{(0)}[8],\ x_4^{(0)}[0..15].$$

Proof. The values of $y_2^{(i)}[8]$ $\forall i = 0, \dots, 15$ can be computed from

$$x_2^{(i)}[8]\ \forall i = 0, \dots, 15$$

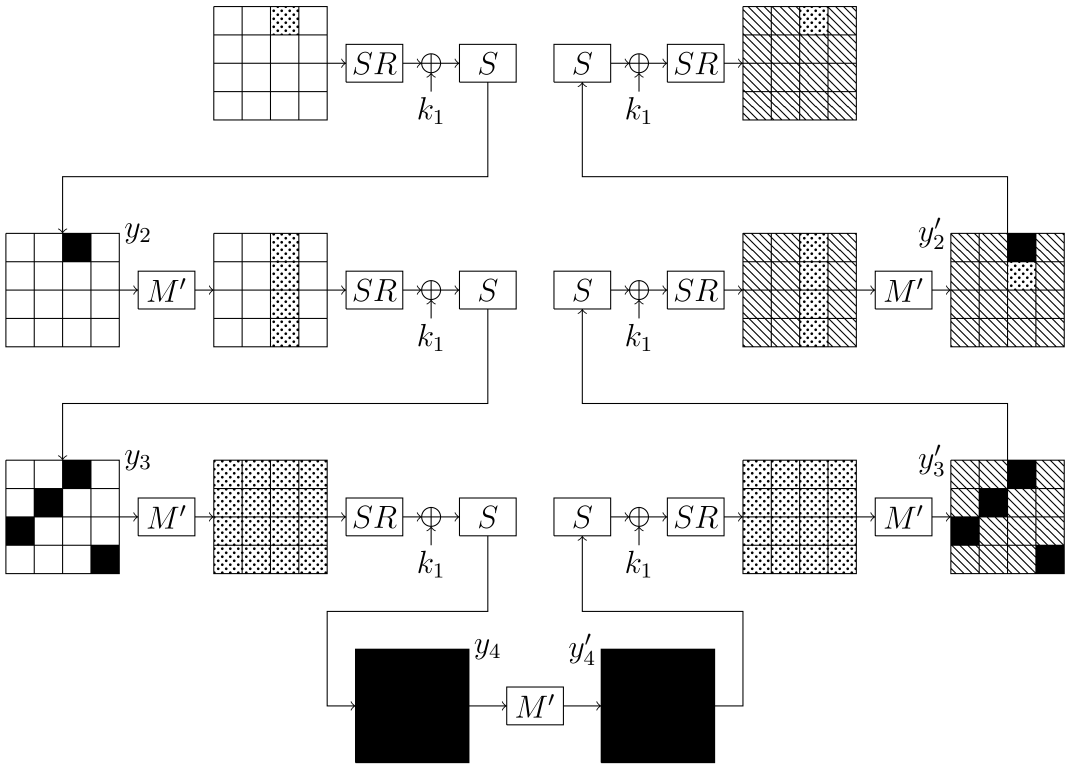


Figure 4.4: The scheme of Observations 8 and 9. White nibbles are constant. Black nibbles serve as parameters. The difference of dotted nibbles is known. Hatched nibbles play no role.

which are known since $x_2^{(0)}[8]$ is a parameter and $\{x_2^{(0)}, \ldots, x_2^{(15)}\}$ is a structured δ-set. The values of

$$x_3^{(i)}[2,5,8,15] \oplus x_3^{(0)}[2,5,8,15] \;\forall i = 1, \ldots, 15$$

can be computed through one M-layer and one key addition layer.

Choosing $x_3^{(0)}[2,5,8,15]$ allows us to compute

$$x_3^{(i)}[2,5,8,15] \;\forall i = 0, \ldots, 15, \;\; y_3^{(i)}[2,5,8,15] \;\forall i = 0, \ldots, 15.$$

Again, the values of $x_4^{(i)} \oplus x_4^{(0)}$ $\forall i = 1, \ldots, 15$ can be computed through one M-layer and one key addition layer.

Choosing $x_4^{(0)}$ allows us to compute $x_4^{(i)}$ $\forall i = 0, \ldots, 15$ and $y_4^{(i)}$ $\forall i = 0, \ldots, 15$. Furthermore, $x'_4 = S^{-1}(M'(y_4))$, so $y_4'^{(i)}$ $\forall i = 0, \ldots, 15$ and $x_4'^{(i)}$ $\forall i = 0, \ldots, 15$ can be computed as well. The values of

$$y_3'^{(i)}[2,5,8,15] \oplus y_3'^{(0)}[2,5,8,15] \;\forall i = 1, \ldots, 15$$

can be computed through one key addition layer and one M-layer.

Choosing $x_3'^{(0)}[2,5,8,15]$ allows us to compute $y_3'^{(0)}[2,5,8,15]$, and in consequence

$$y_3'^{(i)}[2,5,8,15] \;\forall i = 0, \ldots, 15, \;\; x_3'^{(i)}[2,5,8,15] \;\forall i = 0, \ldots, 15.$$

The values of $y_2'^{(i)}[8,9] \oplus y_2'^{(0)}[8,9]$ $\forall i = 1, \ldots, 15$ can be computed through one key addition layer and one M-layer.

Choosing $x_2'^{(0)}[8]$ allows us to compute $y_2'^{(0)}[8]$ and consequently

$$y_2'^{(i)}[8]\ \forall i = 0, \dots, 15,\ x_2'^{(i)}[8]\ \forall i = 0, \dots, 15.$$

□

Observation 9. *The nibble parameters from Observation 8 can be computed from* $x_4^{(0)}[0..15]$ *and* $(M^{-1}(k_1))[2, 5, 8, 12]$. *In other words, there are only* 2^{80} *possible values of the 120-bit sequence from Observation 8.*

Proof. The values of $x_3^{(0)}[2, 5, 8, 15]$ and $x_3'^{(0)}[2, 5, 8, 15]$ can be computed from $(M^{-1}(k_1))[2, 5, 8, 12]$ and $x_4^{(0)}$. The values of $x_2^{(0)}[8]$ and $x_2'^{(0)}[8]$ can be computed from $(M^{-1}(k_1))[8]$, $x_3^{(0)}[2, 5, 8, 15]$ and $x_3'^{(0)}[2, 5, 8, 15]$.

□

Observation 10. *The values of nibbles* $(k_0 \oplus k_1)[8..11], k_1[13]$ *and* $(k_0' \oplus k_1)[8..15]$ *of the PRINCE keys can be computed from the following* 49 *key bits:*

$$(k_0 \oplus k_1)[32..35, 39..47]_b, (k_0' \oplus k_1)[32..36, 40..63]_b, k_1[36..39]_b, k_0[36..38]_b$$

Proof. Thanks to the key schedule algorithm of PRINCE, the value of $k_0'[37..39]_b$ can be computed from $k_0[36..38]_b$. The values of $(k_0 \oplus k_1)[36..38]_b$ and $(k_0' \oplus k_1)[37..39]_b$ can be computed from $k_1[36..39]_b$, $k_0[36..38]_b$ and $k_0'[37..39]_b$.

□

The 8-round attack. The 8-round attack is similar to the 6-round one from Section 4.1. In the offline phase, we compute and store all 2^{80} 120-bit sequences from Observation 9. The online phase goes as follows. We again consider a set of 2^{16} chosen plaintexts with nibbles $8 - 11$ taking all possible values and other nibbles being constant. We encrypt the set through 8-round reduced PRINCE to get 2^{16} ciphertexts. We arbitrarily choose one of the plaintexts to be $p^{(0)}$. We guess several bits of $(k_0 \oplus k_1)$ to obtain a nibble of the state $M(y_1^{(0)})$. We create a structured δ-set $\{M(y_1^{(0)}), \dots, M(y_1^{(15)})\}$ and decrypt the set back to get a set of plaintexts $\{p^{(0)}, \dots, p^{(15)}\}$, a subset of the original 2^{16} chosen plaintexts. We find the corresponding ciphertexts $\{c^{(0)}, \dots, c^{(15)}\}$. We guess several bits of $(k_0' \oplus k_1)$ and k_1 to obtain a nibble of the states $y_2'^{(i)}\ \forall i = 0, \dots, 15$ and a nibble of the states $M(y_1'^{(i)})\ \forall i = 0, \dots, 15$ and compute the sequence

$$[M(y_1'^{(0)})[8] \oplus M(y_1'^{(1)})[8], \dots, M(y_1'^{(0)})[8] \oplus M(y_1'^{(15)})[8],$$
$$y_2'^{(0)}[9] \oplus y_2'^{(1)}[9], \dots, y_2'^{(0)}[9] \oplus y_2'^{(15)}[9]].$$

If there is such sequence among the 2^{80} possible sequences stored from the offline phase, then the guessed key bits are candidates for the actual key bits. The attack is depicted in Figure 4.5 and Algorithm 7 gives its pseudocode.

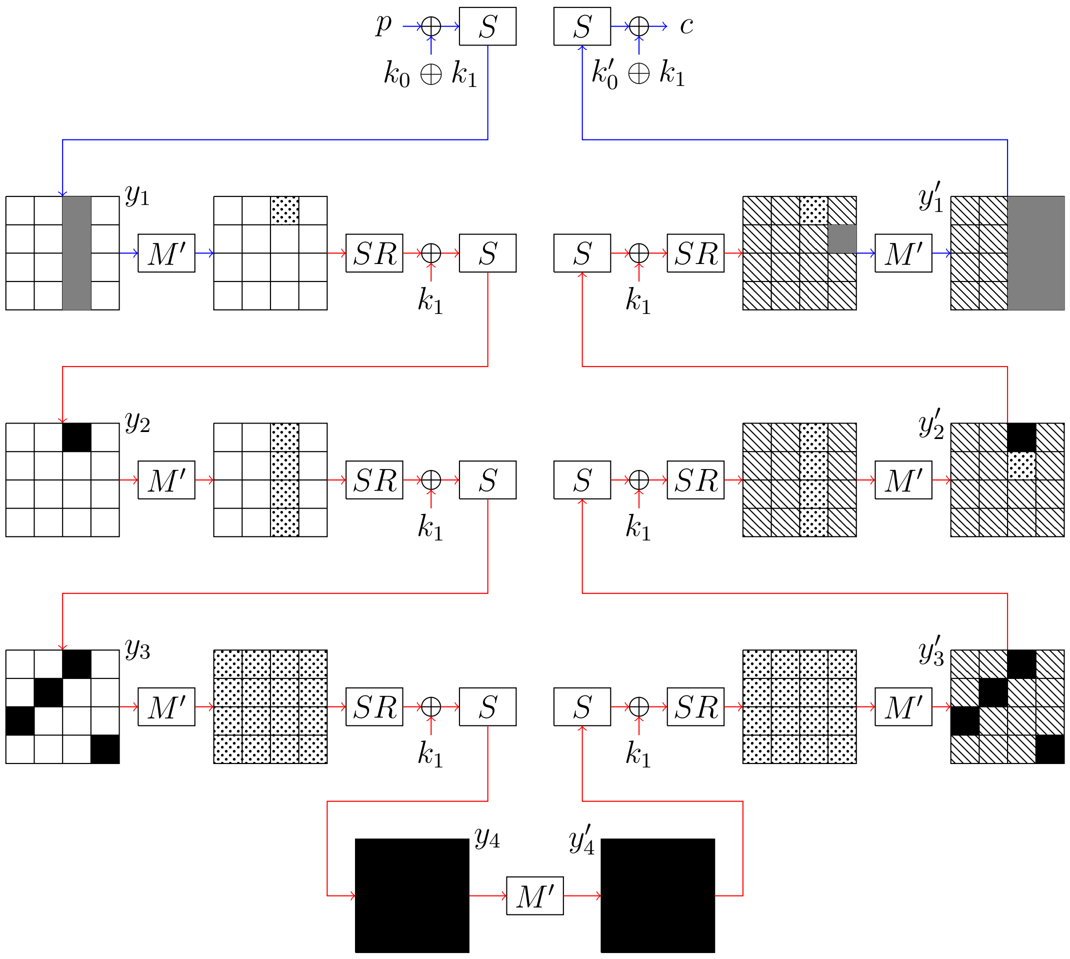


Figure 4.5: The 8-round meet-in-the-middle attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

Again, let us take a look at the complexity of the attack. First, the probability of an incorrect guess to pass the meet-in-the-middle criterion is $2^{80} \times 2^{-120} = 2^{-40}$. Since we guess 2^{49} key bits, we expect to get 2^9 key candidates after one iteration of the attack and only one key candidate after two iterations of the attack.

The data complexity is 2^{16} chosen plaintexts. The memory requirements are $2^{80} \times 30 \times 4$ bits to store the sequences from the offline phase and $2^{16} \times 64$ bits to store the ciphertexts during the online phase. The time complexity is 2^{16} 8-round PRINCE encryptions for producing ciphertexts plus $(2^{49} + 2^9) \times 16$ encryptions through 13 S-boxes for creating the δ-sets and the states $M(y'_1)$ and y'_2. Altogether, it is approximately $2^{49.7}$ 8-round PRINCE encryptions. The complexity of the online phase is computing the 2^{80} possible sequences. This requires encryption through

$$2^4 \times 16 + 2^{16} \times 16 \times 4 + 2 \times 2^{80} \times 16 \times 16 + 2^{80} \times 16 \times 4 + 2^{80} \times 16$$

S-boxes, which is approximately $2^{82.2}$ 8-round PRINCE encryptions. To this date, this is arguably the most practical attack on 8-rounds of PRINCE.

Algorithm 7 The 8-round meet-in-the-middle attack

```
1: Precompute the set $S$ of all $2^{80}$ possible sequences of
       $$[M(y_1'^{(0)})[8] \oplus M(y_1'^{(1)})[8], \dots, M(y_1'^{(0)})[8] \oplus M(y_1'^{(15)})[8], y_2'^{(0)}[9] \oplus y_2'^{(1)}[9], \dots, y_2'^{(0)}[9] \oplus y_2'^{(15)}[9]].$$
2: Set $P$ a set of $2^{16}$ chosen plaintexts such that nibbles $8 - 11$ take all possible values and all other nibbles are constant.
3: Encrypt all plaintexts in $P$ to get $2^{16}$ ciphertexts.
4: Arbitrarily choose a plaintext $p^{(0)} \in P$.
5: for all $2^{49}$ values of $(k_0 \oplus k_1)[8..11]$, $(k_0' \oplus k_1)[8..15]$ and $k_1[13]$ do
6:     Find $M(y_1^{(0)})[8]$.
7:     For $i = 1, \dots, 15$ set $M(y_1^{(i)})[8]$ so that $M(y_1^{(i)})[8] \oplus M(y_1^{(0)})[8] = i$.
8:     Find $p^{(i)}$ $\forall i = 1, \dots, 15$.
9:     Find the corresponding ciphertexts $c^{(i)}$ $\forall i = 0, \dots, 15$.
10:    Find $M(y_1'^{(i)})[8]$ $\forall i = 0, \dots, 15$.
11:    Find $y_2'^{(i)}[9]$ $\forall i = 0, \dots, 15$.
12:    Compute the sequence
       $$s = [M(y_1'^{(0)})[8] \oplus M(y_1'^{(1)})[8], \dots, M(y_1'^{(0)})[8] \oplus M(y_1'^{(15)})[8], y_2'^{(0)}[9] \oplus y_2'^{(1)}[9], \dots, y_2'^{(0)}[9] \oplus y_2'^{(15)}[9]].$$
13:    If $s \notin S$, then the guessed value is not a key candidate.
14: end for
15: Repeat Steps 4 through 14 until there is only one key candidate.
```

Note that this attack only recovers 49 out of 128 key bits. To recover the remaining key bits, we can run different variants of this attack which share the same idea as the attack studied in this section but consider different nibbles. These additional attacks do not significantly increase the time or the data complexity. An example scheme of such an attack can be found in Attachment A.4.

4.3 7-round meet-in-the-middle attack

The paper [5] does not consider any attacks on 7 rounds of PRINCE. This might be because there was no such category in the original PRINCE challenge. However, there does not seem to be any attack on 7 rounds of PRINCE meeting the criterion of only 2^{20} chosen plaintexts (this was the requirement for all existing categories in the challenge) studied so far. The current most practical 7-round attacks require slightly above 2^{32} chosen plaintexts.

This section shows a way to build a 7-round meet-in-the-middle attack from the 8-round one mentioned in the previous section and proposes a new 7-round meet-in-the-middle-attack with lower time complexity of the offline phase. Both of these attacks only need 2^{16} chosen plaintexts.

Reducing the 8-round attack. The first, rather obvious, way to create a 7-round attack is to use the 8-round attack from the previous section and omit the eighth round of the cipher. The key observation would be as follows.

Consider a set of 16 plaintexts $\{p^{(0)}, \dots, p^{(15)}\}$ and its encryption through 7-round reduced PRINCE. If the set $\{M(y_1^{(0)}), \dots, M(y_1^{(15)})\}$ is a structured δ-set, then the 120-bit sequence

$$[y_1'^{(1)}[8] \oplus y_1'^{(0)}[8], \dots, y_1'^{(15)}[8] \oplus y_1'^{(0)}[8], y_1'^{(1)}[9] \oplus y_1'^{(0)}[9], \dots, y_1'^{(15)}[9] \oplus y_1'^{(0)}[9]]$$

only depends on 80 bits.

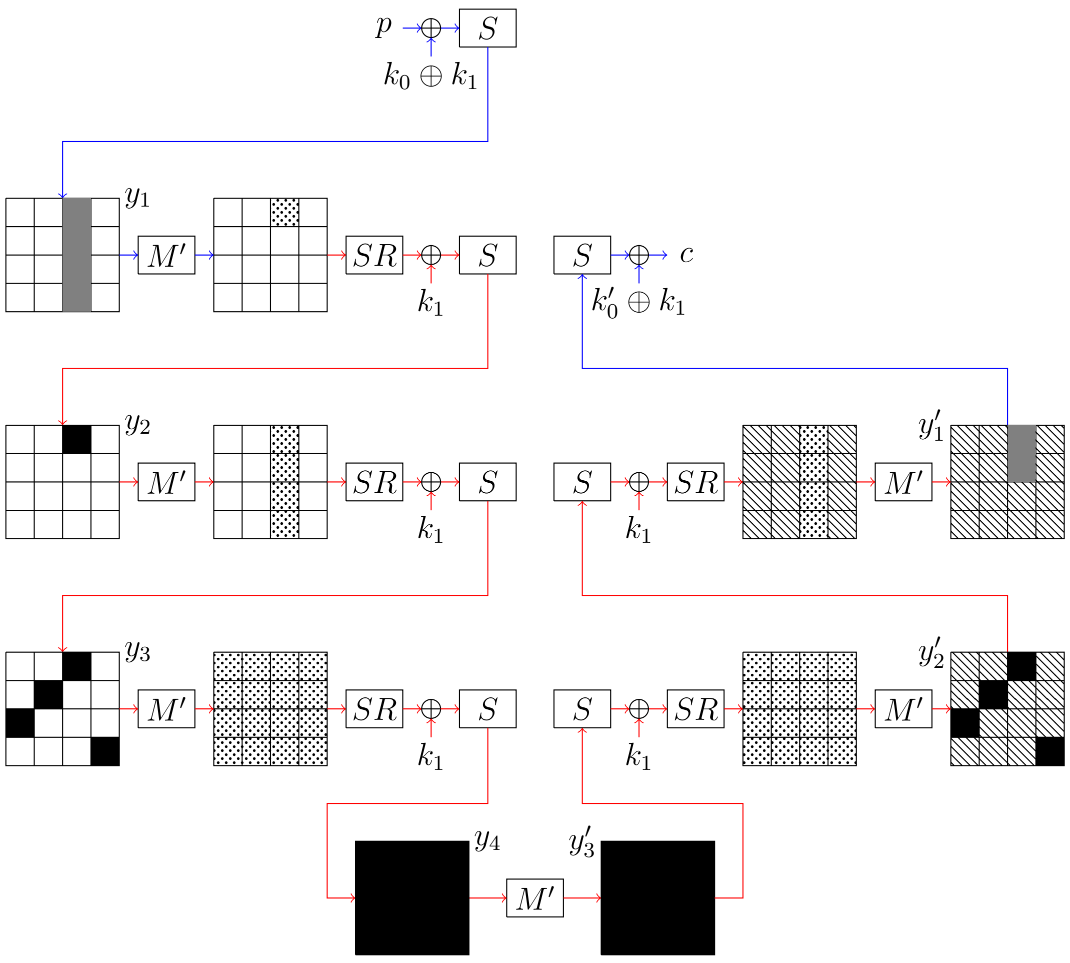


Figure 4.6: The 7-round meet-in-the-middle attack derived from the original 8-round attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

The online phase of the attack would require guessing four nibbles of the key $k_0 \oplus k_1$ and two nibbles of the key $k_0' \oplus k_1$. Therefore, 24 bits of the whole key would be recovered with a time complexity of approximately $2^{23.8}$ 7-round PRINCE encryptions. The time complexity of the offline phase would be encrypting through $2^4 \times 16 + 2^{16} \times 16 \times 4 + 2 \times 2^{80} \times 16 \times 16 + 2^{80} \times 16 \times 4$ S-boxes, which is approximately $2^{82.4}$ 7-round PRINCE encryptions. To recover the remaining

Algorithm 8 The basic 7-round meet-in-the-middle attack

```
1: Precompute the set $S$ of all $2^{80}$ possible sequences of
   $$[y_1'^{(0)}[8] \oplus y_1'^{(1)}[8], \dots, y_1'^{(0)}[8] \oplus y_1'^{(15)}[8], y_1'^{(0)}[9] \oplus y_1'^{(1)}[9], \dots, y_1'^{(0)}[9] \oplus y_1'^{(15)}[9]].$$
2: Set $P$ a set of $2^{16}$ chosen plaintexts such that nibbles $8-11$ take all possible values and all other nibbles are constant.
3: Encrypt all plaintexts in $P$ to get $2^{16}$ ciphertexts.
4: Arbitrarily choose a plaintext $p^{(0)} \in P$.
5: for all $2^{24}$ values of $(k_0 \oplus k_1)[8..11]$ and $(k_0' \oplus k_1)[8..9]$ do
6:     Find $M(y_1^{(0)})[8]$.
7:     For $i = 1, \dots, 15$ set $M(y_1^{(i)})[8]$ so that $M(y_1^{(i)})[8] \oplus M(y_1^{(0)})[8] = i$.
8:     Find $p^{(i)}\ \forall i = 1, \dots, 15$.
9:     Find the corresponding ciphertexts $c^{(i)}\ \forall i = 0, \dots, 15$.
10:    Find $y_1'^{(i)}[8]\ \forall i = 0, \dots, 15$.
11:    Find $y_1'^{(i)}[9]\ \forall i = 0, \dots, 15$.
12:    Compute the sequence
       $$s = [y_1'^{(0)}[8] \oplus y_1'^{(1)}[8], \dots, y_1'^{(0)}[8] \oplus y_1'^{(15)}[8], y_1'^{(0)}[9] \oplus y_1'^{(1)}[9], \dots, y_1'^{(0)}[9] \oplus y_1'^{(15)}[9]].$$
13:    If $s \notin S$, then the guessed value is not a key candidate.
14: end for
15: Repeat Steps 4 through 14 until there is only one key candidate.
```

key bits, additional attacks would have to be performed. Both the offline phase and the online phase of this attack are depicted in Figure 4.6. A pseudocode is given in Algorithm 8.

The new 7-round attack. Another way to create a 7-round attack is to find a different meet-in-the-middle criterion. This new attack uses the same idea as the previous ones however it presents a new meet-in-the-middle-criterion which focuses on differences in individual bits rather than on differences of nibbles. The meet-in-the-middle criterion used in the offline phase of the attack is presented in Observation 11 and the number of key bits guessed during the online phase is discussed in Observation 12.

Observation 11. *Consider a set of* 16 *plaintexts* $\{p^{(0)}, \dots, p^{(15)}\}$ *and its encryption through 7-round reduced PRINCE. If the set* $\{y_2^{(0)}, \dots, y_2^{(15)}\}$ *is a structured* δ*-set, then the sequence*

$$[y_2'^{(1)}[11,20,32,42,63]_b \oplus y_2'^{(0)}[11,20,32,42,63]_b, \dots, y_2'^{(15)}[11,20,32,42,63]_b \oplus y_2'^{(0)}[11,20,32,42,63]_b]$$

is fully determined by nibbles $x_3^{(0)}[2,5,8,15]$ *and* $x_4^{(0)}[0..11]$.

In other words, the 75*-bit sequence can only have* 2^{64} *values.*

Proof. The proof is similar to the proofs of Observations 6 and 8. The only step worth mentioning is the transition from x'_3 to y'_2. Just before the M' layer, we know the differences in 3 nibbles in each column. Recall the bitwise formulas for M' (see 3.1) from Section 3.1. The knowledge of three nibbles of one column allows us to compute exactly one bit of each nibble through the M' layer. $\square$

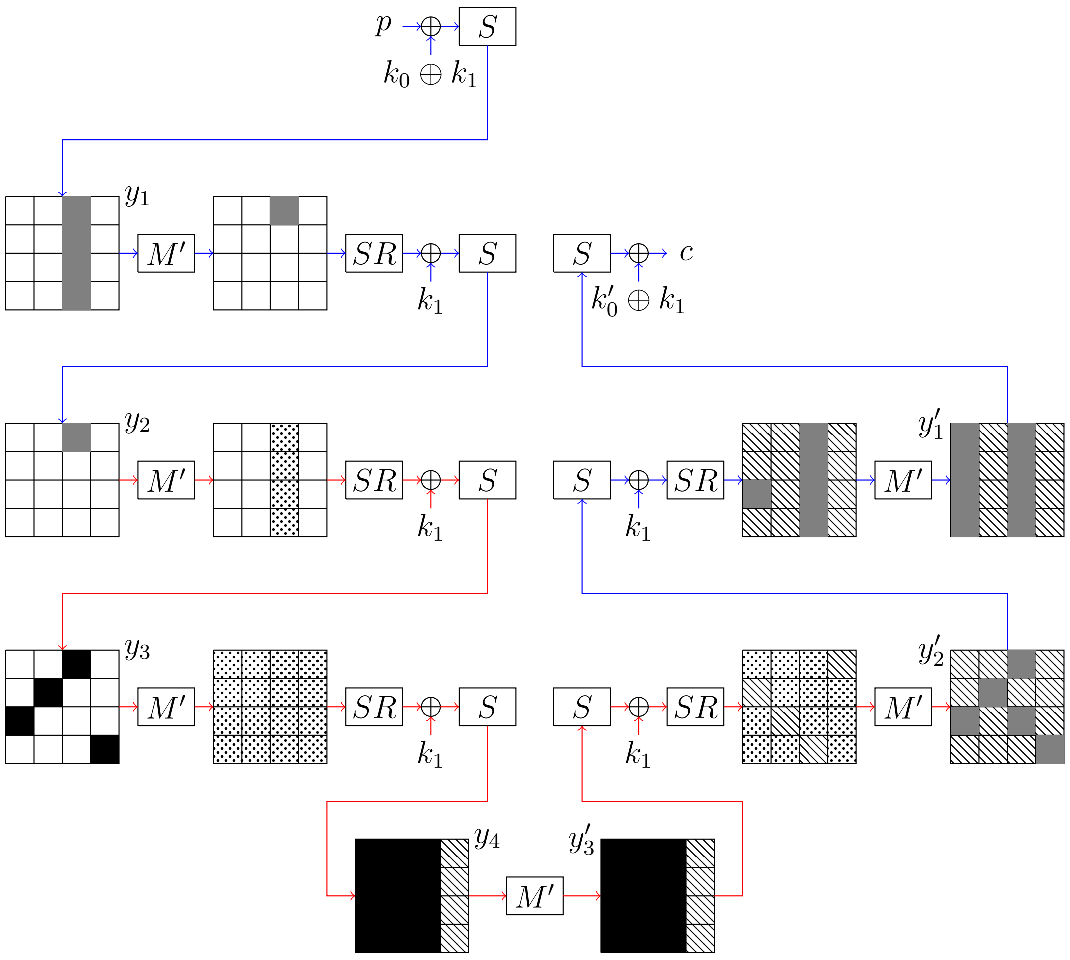


Figure 4.7: The new 7-round meet-in-the-middle attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

Observation 12. *The values of nibbles* $(k_0 \oplus k_1)[8..11], k_1[2,5,8,10,15]$ *and* $(k'_0 \oplus k_1)[0..3, 8..11]$ *of the PRINCE keys can be computed from the following* 61 *key bits:*

$$(k_0 \oplus k_1)[36..39, 44..47]_b, (k'_0 \oplus k_1)[0..15, 36..39, 44..47]_b,$$
$$k_1[8..11, 20..23, 32..35, 40..43, 60..63]_b, k_0[31..35, 40..43]_b$$

Proof. Thanks to the key schedule algorithm, the value of $k'_0[32..35, 40..43]_b$ can be computed from $k_0[31..35, 39..43]_b$. The value of $k_0[39]_b$ can be computed from $k_0[35]_b$, $(k_0 \oplus k_1)[36..39]_b$ and $(k'_0 \oplus k_1)[36..39]_b$.

The values of $(k_0 \oplus k_1)[31..35, 40..43]_b$ and $(k'_0 \oplus k_1)[31..35, 40..43]_b$ can be computed from $k_1[31..35, 40..43]_b$, $k_0[31..35, 40..43]_b$ and $k'_0[31..35, 40..43]_b$.

□

The attack, depicted in Figure 4.7, follows the same strategy as the previous meet-in-the-middle attacks. We precompute the 2^{64} 75-bit sequences from Observation 11 in the offline phase, we guess the 61 key bits from Observation 12 in the online phase and we repeat the attack until there is only one key candidate left. The pseudocode of the attack is given in Algorithm 9.

Algorithm 9 The new 7-round meet-in-the-middle attack

```
1: Precompute the set $S$ of all $2^{64}$ possible sequences of
       $[y_2'^{(0)}[11,20,32,42,60]_b \oplus y_2'^{(1)}[11,20,32,42,60]_b, \ldots, y_2'^{(0)}[11,20,32,42,60]_b \oplus y_2'^{(15)}[11,20,32,42,60]_b]$.
2: Set $P$ a set of $2^{16}$ chosen plaintexts such that nibbles $8-11$ take all possible values and all other nibbles are constant.
3: Encrypt all plaintexts in $P$ to get $2^{16}$ ciphertexts.
4: Arbitrarily choose a plaintext $p^{(0)} \in P$.
5: **for all** $2^{62}$ values of $(k_0 \oplus k_1)[8..11]$, $(k'_0 \oplus k_1)[0..3, 8..11]$ and $k_1[2,5,8,10,15]$ **do**
6:     Find $y_2^{(0)}[8]$.
7:     For $i = 1, \ldots, 15$ set $y_2^{(i)}[8]$ so that $y_2^{(i)}[8] \oplus y_2^{(0)}[8] = i$.
8:     Find $p^{(i)}$ $\forall i = 1, \ldots, 15$.
9:     Find the corresponding ciphertexts $c^{(i)}$ $\forall i = 0, \ldots, 15$.
10:    Find $y_2'^{(i)}[11,20,32,42,60]_b$ $\forall i = 0, \ldots, 15$.
11:    Compute the sequence
       $s = [y_2'^{(0)}[11,20,32,42,60]_b \oplus y_2'^{(1)}[11,20,32,42,60]_b, \ldots, y_2'^{(0)}[11,20,32,42,60]_b \oplus y_2'^{(15)}[11,20,32,42,60]_b]$.
12:    If $s \notin S$, then the guessed value is not a key candidate.
13: **end for**
14: Repeat Steps 4 through 13 until there is only one key candidate.
```

One round of the attack is expected to reduce the number of key candidates by a factor of $2^{75} \times 2^{-64} = 2^{11}$, so several iterations must be performed. The time complexity of the offline phase is encrypting through $2^{16} \times 16 \times 4 + 2 \times 2^{64} \times 16 \times 12$ S-boxes, which is approximately $2^{65.8}$ 7-round PRINCE encryptions. The time complexity of the online phase of the attack is $(2^{61} + 2^{50} + 2^{39} + 2^{28} + 2^{17} + 2^{6}) \times 16$ encryptions through 18 S-boxes, that is approximately $2^{62.4}$ 7-round PRINCE encryptions. As we have already mentioned, the data complexity is only 2^{16} chosen plaintexts. The memory required is $2^{64} \times 75$ bits to store the sequence from the offline phase and $2^{16} \times 64$ bits to store the ciphertexts during the online phase.

The attack recovers 61 key bits. To get the remaining bits, one option is to continue by guessing one additional nibble (one of nibbles $0, 7, 13$) of the key k_1 each time. Note that the sequence from the offline phase would have to be extended for $2^{64} \times 15$ bits to cover the corresponding bit of y_2' as well. Finding the key nibbles $k_1[0, 7, 13]$ is enough for us to be able to perform an exhaustive search to find the remaining key bits as only 14 key nibbles are left unknown.

As we can see, the first 7-round attack has an offline phase with time complexity of $2^{82.4}$ and an online phase with time complexity of $2^{23.8}$ 7-round PRINCE encryptions. This new attack has an offline phase with time complexity of $2^{65.8}$ and an online phase with time complexity of $2^{62.4}$ 7-round PRINCE encryptions. Both of them require only 2^{16} chosen plaintexts. In comparison, the current most practical attack on 7 rounds of PRINCE, a higher order differential attack presented in [4], has a time complexity of $2^{44.3}$ 7-round PRINCE encryptions but needs 2^{33} chosen plaintexts.

Conclusion

In this work, we have studied the current best attacks on round reduced versions of the cipher PRINCE in detail. We have provided our own Python 3 implementations of PRINCE and of the integral attacks on four and five rounds of the cipher. We have presented a theoretical proof of the 3.5-round integral distinguisher, which was omitted in the original paper, and we have presented a new 4.5-round integral distinguisher, allowing more efficient attacks on five and six rounds of the cipher. We have also proposed a new meet-in-the-middle attack on 7 rounds of PRINCE with low data complexity.

Bibliography

[1] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. *Advances in Cryptology – ASIACRYPT 2012*, pages 208–225, 2012.

[2] Paweł Morawiecki. Practical attacks on the round-reduced PRINCE. *IACR Cryptol. ePrint Arch.*, 2015:245, 2017.

[3] Raluca Posteuca and Gabriel Negara. Integral cryptanalysis of round-reduced PRINCE cipher. *Proceedings of the Romanian Academy - Series A: Mathematics, Physics, Technical Sciences, Information Science*, 16:265–269, 01 2015.

[4] Shahram Rasoolzadeh and Håvard Raddum. Faster Key Recovery Attack on Round-Reduced PRINCE. In Andrey Bogdanov, editor, *Lightweight Cryptography for Security and Privacy - 5th International Workshop, LightSec 2016, Aksaray, Turkey, September 21-22, 2016, Revised Selected Papers*, volume 10098 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2016.

[5] Patrick Derbez and Léo Perrin. Meet-in-the-Middle Attacks and Structural Analysis of Round-Reduced PRINCE. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 190–216. Springer, 2015.

[6] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[7] Lars Ramkilde Knudsen and David A. Wagner. Integral Cryptanalysis (extended abstract). In *FSE 2002*, 2002.

[8] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher SQUARE. *Lecture Notes in Computer Science*, 1267, 10 1998.

[9] W. Diffie and M.E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.

List of Figures

1.1 A typical key addition layer 5
1.2 A scheme of a substitution layer 5
1.3 A scheme of a permutation layer 5
1.4 Multiple rounds of SPN 6
1.5 A scheme of the FX construction 7
1.6 An example of an integral attack 9
1.7 An example of a meet-in-the-middle attack 10

2.1 The scheme of PRINCE 11
2.2 The scheme of PRINCE$_{\text{core}}$ 12
2.3 The PRINCE S-box 12
2.4 The round constants of PRINCE 13
2.5 The shift rows operation 13

3.1 Arranging the PRINCE state into a square scheme 16
3.2 The scheme of the 3.5-round integral distinguisher 19
3.3 The scheme of the 2.5-round integral distinguisher 21
3.4 The scheme of the original 4.5-round integral distinguisher 22
3.5 The first two states of the new 4.5-round integral distinguisher . . 23

4.1 The notation of the PRINCE states 30
4.2 The scheme of Observation 6 31
4.3 The 6-round meet-in-the-middle attack 32
4.4 The scheme of Observations 8 and 9 34
4.5 The 8-round meet-in-the-middle attack 36
4.6 The 7-round meet-in-the-middle attack derived from the original 8-round attack 38
4.7 The new 7-round meet-in-the-middle attack 40

A.1 The alternative 6-round meet-in-the-middle attack 48
A.2 The additional 6-round meet-in-the-middle attack 49
A.3 The alternative 8-round meet-in-the-middle attack 50

List of Algorithms

1 The basic 4-round integral attack 20
2 The basic 5-round integral attack 24
3 The full 4-round integral attack 25
4 The full 5-round integral attack 27
5 The full 6-round integral attack 28
6 The 6-round meet-in-the-middle attack 33
7 The 8-round meet-in-the-middle attack 37
8 The basic 7-round meet-in-the-middle attack 39
9 The new 7-round meet-in-the-middle attack 41

A. Attachments

A.1 Attachment one - PRINCE Python 3 implementation

The source code of our Python 3 reference implementation of PRINCE is available at `https://github.com/DavidTvrdy/PRINCE/blob/main/Prince.py`. The implementation works on the test vectors provided by the authors of the cipher in [1]. It is a straightforward, unoptimized implementation. No effort has been made to defend against side-channel or any other kind of attacks.

The function `Encrypt(key, message)` is used for encryption. The function `Decrypt(key, message)` is used for decryption. To check the test vectors provided in [1], use the function `Test()`.

The instructions and comments are included in the file itself as well.

A.2 Attachment two - Python 3 implementation of round reduced integral attacks

The source code of our Python 3 implementation of the integral attacks on round reduced version of PRINCE is available at `https://github.com/DavidTvrdy/PRINCE/blob/main/Integral_attacks_on_prince.py`.

The file is divided into several sections. The first section contains the reference implementation of PRINCE. Each one of the remaining sections provides one attack on round reduced version of PRINCE.

The attack in the `Square4BasicSingle(secret_key)` function performs the basic attack from Section 3.1, Algorithm 1 and recovers one nibble of the key ($k_0' \oplus k_1$) in the case of four rounds.

The attack in the `Square4BasicFull(secret_key)` function performs the basic attack from Section 3.1 and recovers the whole key k_0, k_1 in the case of four rounds.

The attack in the `Square4ArraysFull(secret_key)` function performs the full attack from Section 3.3, Algorithm 3 and recovers the whole key k_0, k_1 in the case of four rounds.

The attack in the `Square5ArraysFull(secret_key)` function performs the full attack on five rounds of the cipher and recovers the whole key k_0, k_1. In this case, the original distinguisher from Sec. 3.2.1 is used.

The attack in the `Square5ArraysFullNew(secret_key)` function performs the full attack on five rounds of PRINCE and recovers the whole key k_0, k_1. In this case, the new distinguisher from Sec. 3.2.2 is used.

The instructions and comments are included in the file itself as well.

A.3 Attachment three - Recovery of the remaining key bits after the 6-round meet-in-the-middle attack

The 6-round meet-in-the-middle attack from Section 4.1 recovers 33 key bits. This section shows a possible way to recover the remaining key bits. Figure A.1 shows a similar attack which shares the same idea but recovers different key nibbles, specifically the bits $(k'_0 \oplus k_1)[0..15]_b$ and $k_1[40..43]_b$. The time complexity of this attack is indeed lower than the complexity of the attack presented in Section 4.1 as only 16 unknown key bits are guessed this time. Note that the whole column $k_1[8..11]$ can be derived from $k_1[8]$, $(k_0 \oplus k_1)[8..11]$ and $(k'_0 \oplus k_1)[8..11]$, which are already known. The complexity of the offline phase remains the same however the meet-in-the-middle sequence computed is different.

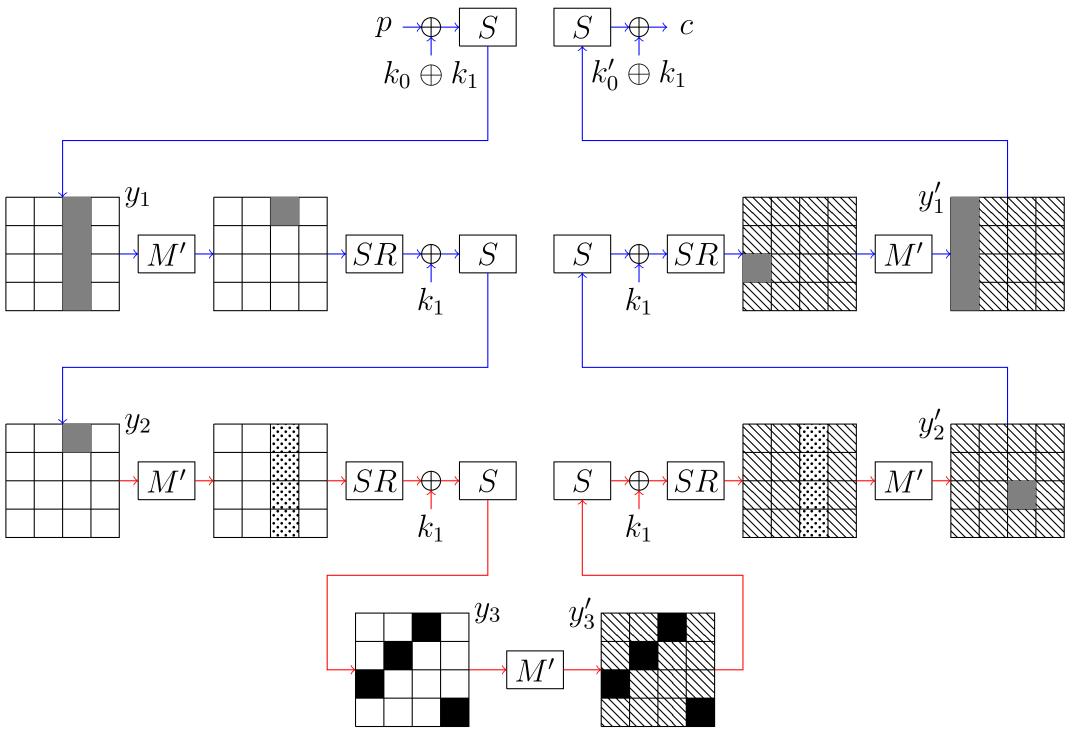


Figure A.1: The alternative 6-round meet-in-the-middle attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

The alternative attack can be performed in different variants. One of them is guessing the bits $(k'_0 \oplus k_1)[16..31]_b$ and $k_1[44..47]_b$. Another one is guessing the bits $(k'_0 \oplus k_1)[48..63]_b$ and $k_1[36..39]_b$. Each of the variants mentioned recovers at most 16 additional key bits. If we use all of them, the whole key $(k'_0 \oplus k_1)$ is recovered.

A column of the key k_1 can be recovered using a different attack that uses the knowledge of the whole key $(k'_0 \oplus k_1)$. The idea of the attack is again the same, the scheme is depicted in Figure A.2. The complexity of the additional attack is indeed lower than the complexity of recovering the previous key bits.

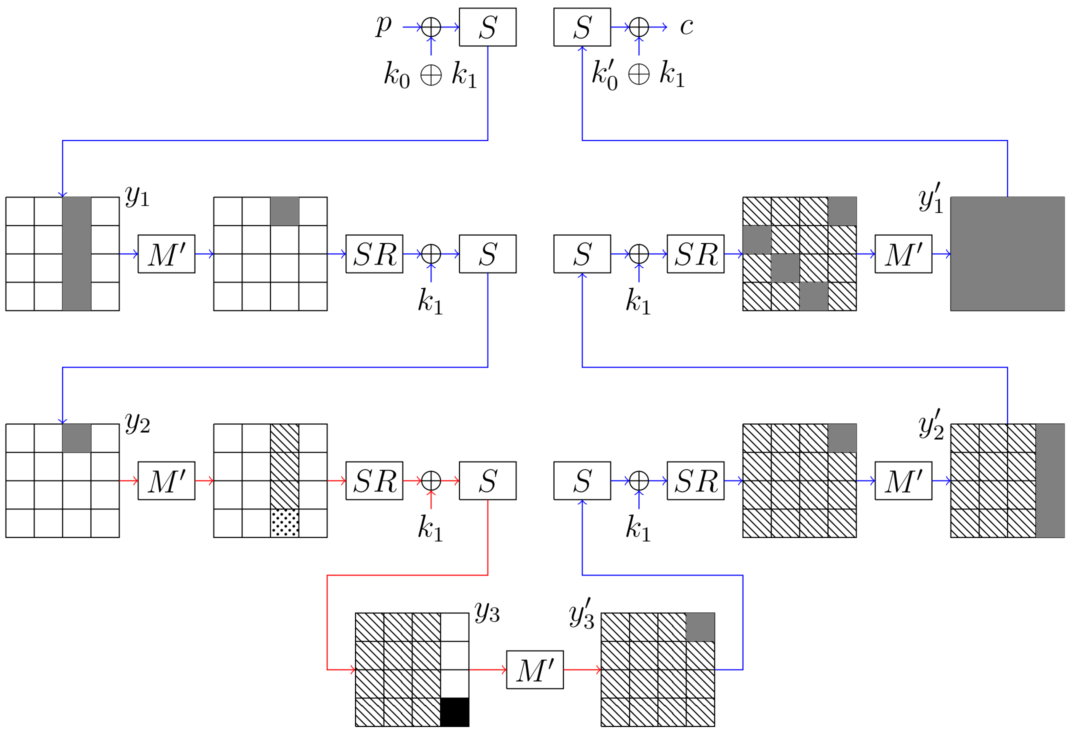


Figure A.2: The additional 6-round meet-in-the-middle attack. Note that the whole key $k'_0 \oplus k_1$ is known. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

After performing such an attack, only two columns of k_1 are left to find. We can recover them by performing an exhaustive search without increasing the overall complexity.

A.4 Attachment four - Recovery of the remaining key bits after the 8-round meet-in-the-middle attack

The 8-round meet-in-the-middle attack from Section 4.2 recovers 49 key bits. This section shows how to recover the remaining key bits. The attack and the graphics of the figure is taken from [5]. Figure A.3 shows a similar attack which shares the same idea but recovers different key nibbles, specifically the bits $(k'_0 \oplus k_1)[0..31]_b$ and $k_1[40..47]_b$. The time complexity of this attack is indeed lower than the complexity of the attack presented in Section 4.2 as only 33 unknown key bits are guessed this time. Note that $k_1[32..47]_b$ and $k_0[32..47]_b$ can be computed from $k_1[47]_b$ since the whole key $k'_0 \oplus k_1$ and $(k_0 \oplus k_1)[32..47]$ have already been guessed or are known. The complexity of the offline phase remains the same however the meet-in-the-middle sequence computed is different.

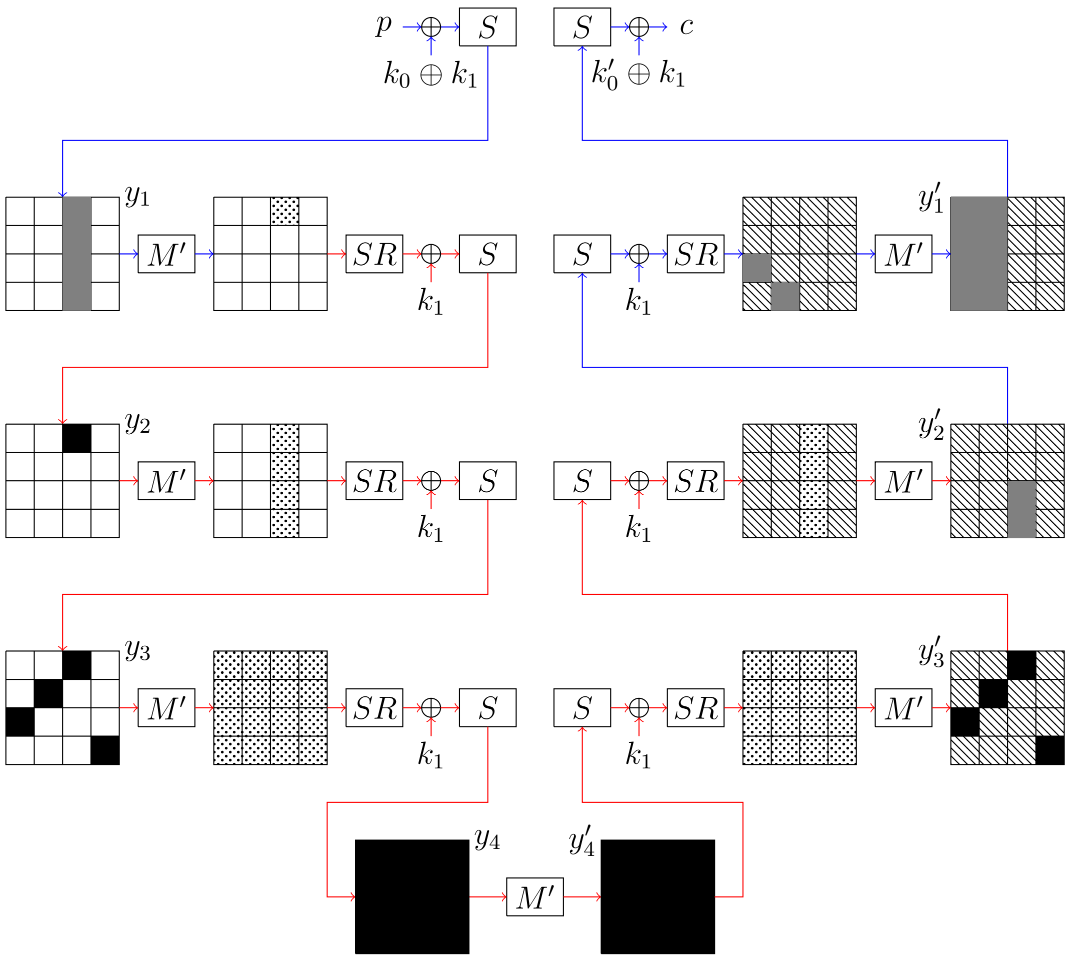


Figure A.3: The alternative 8-round meet-in-the-middle attack. Blue parts are connected to the online phase, red parts are connected to the offline phase. Gray nibbles need to be guessed during the online phase. Black nibbles need to be guessed during the offline phase. White nibbles are constant. The difference in dotted nibbles is known. Hatched nibbles play no role.

After performing the above mentioned attack, only nibbles $k_1[0..7, 12..15]$ need to be recovered. These key bits can be found by exhaustive search without increasing the overall time complexity.