Daniel Kales, BSc

Cryptanalysis of Tweakable Block Ciphers

Application to MANTIS and QARMA

Master's Thesis

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to
Graz University of Technology

Supervisor
Dipl.-Ing. Maria Eichlseder

Univ.-Prof. Dipl.-Ing. Dr.techn. Christian Rechberger

Institute of Applied Information Processing and Communications

Faculty of Computer Science and Biomedical Engineering

Graz, October 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Cryptography is the science of secure communication over a possibly insecure channel. One of the classical fundamental building blocks is the block cipher, which maps a plaintext to an encrypted ciphertext using a transformation selected by a secret key. Tweakable block ciphers (TBCs) are a modern cryptographic primitive that is better suited to protect both the confidentiality and authenticity of long messages. However, their security is not as well understood as classical block ciphers. In contrast to classical block ciphers, we also need to consider so-called "related-tweak" attacks where the attacker has full control over the tweak input. In this thesis we propose several improvements to existing differential attacks on tweakable block ciphers and apply these improvements to the lightweight tweakable block cipher MANTIS. We provide a practical implementation for an existing key-recovery attack on MANTIS_5 and use the improved methodology to improve its data and time complexity. Furthermore, we propose a new key-recovery attack on MANTIS_6, which can recover the secret key while staying below the general security limit of 2^n data and 2^{126-n} time complexity. Finally, we try to apply the ideas of the attacks to other modern tweakable block ciphers and observe the implications of the different cipher structures on the practicability of the attack.

Kurzfassung

Kryptografie ist die Wissenschaft der sicheren Kommunikation über einen möglicherweise unsicheren Kanal. Ein grundlegender Baustein der Kryptografie ist die Blockchiffre, die einen Klartext auf einen verschlüsselten Geheimtext abbildet, wobei ein geheimer Schlüssel die verwendete Transformation definiert. „Tweakable“ Blockchiffren (TBCs) sind moderne kryptografische Algorithmen, die besser dafür geeignet sind, sowohl die Vertraulichkeit als auch die Authentizität von langen Nachrichten zu beschützen. Jedoch ist die Sicherheit von TBCs bisher weniger gut untersucht als die von klassischen Blockchiffren. Im Gegensatz zu klassischen Blockchiffren sind auch noch sogenannte „related-tweak“ Angriffe zu berücksichtigen, bei denen ein Angreifer volle Kontrolle über den zusätzlichen Tweak-Parameter hat. In dieser Arbeit stellen wir einige Verbesserungen zu bestehenden differentiellen Angriffen auf TBCs vor und wenden diese auf MANTIS an, einen kürzlich publizierten TBC, der besonders für Anwendungen mit strikten Latenzanforderungen optimiert ist. Wir stellen eine praktische Implementierung für einen bestehenden Angriff auf MANTIS_5 vor und benutzen unsere neuen Methoden, um die Daten- und Zeitkomplexität des Angriffs zu verbessern. Des Weiteren stellen wir einen neuen Angriff auf MANTIS_6 vor, der den geheimen Schlüssel schneller findet als die allgemeinen Sicherheitsschranken von 2^n Datenkomplexität und 2^{126-n} Zeitkomplexität. Schlussendlich wenden wir die Ideen des Angriffs auf andere moderne TBCs an und beobachten die Einflüsse der unterschiedlichen internen Strukturen der Chiffren auf die Ergebnisse des Angriffs.

Contents

List of Figures

List of Tables

1 Introduction and State of the Art

Block ciphers are one of the most fundamental building blocks in cryptography. They are used in various constructions to provide encryption and/or authentication. Both Feistel as well as Substitution-Permutation primitives have been the focus point of extensive research over the last few decades and their problems (especially against linear and differential cryptanalysis) are now well understood. Basic block ciphers (which are essentially pseudo-random permutations) are deterministic by design. They take a key k and a message m and output a ciphertext c and as long as the key and message are the same, so is the result of the encryption. However, this is a problem for security. When using an encryption, we do not want to be able to tell whether the same message is sent twice. Therefore, many higher-level modes of operation that use block ciphers as a basic building block add a so-called "nonce" as a further input. This nonce ("number only used once") is a number that is unique for each encryption, which is added to the input or the output of the block cipher, giving it a non-deterministic behavior. Liskov et al. [LRW02] aimed to bring this often-used primitive down to the block cipher level and proposed the concept of "Tweakable Block Ciphers" (TBCs).

The security of tweakable block ciphers is not as well understood as the security of classic block ciphers. That is why we investigate the differential behavior of modern tweakable block ciphers.

1.1 Block Ciphers

Block ciphers are one of the oldest symmetric cryptographic primitives and can be used to guarantee confidentiality of data. A block cipher is a cryptographic algorithm operating on a fixed number of bits (called block). The secret key

selects the transformation the algorithm is performing and the result is deterministic when using the same key. Formally, a block cipher is specified by its encryption function

$$E(K, P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n,$$

where K is the secret key of size k bits, P is the plaintext (often also called message M) of size n, where n is the block size. The result of the operation is the ciphertext C, which is also of length n. The encryption function has an inverse function, the decryption function

$$D(K, C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n,$$

which takes a key K and the ciphertext C and returns the original plaintext P.

Two of the most widely known algorithms are the Data Encryption Standard (DES) [Cop94] and the Advanced Encryption Standard (AES) [DR02]. Two of the main design approaches for block ciphers are Feistel networks (for example DES) and Substitution-Permutation networks (for example AES). The security of both of these constructions is a topic that was researched intensely during the last four decades and is well understood today.

Two of the main attack methods for block ciphers are linear and differential cryptanalysis. In the following section we will describe differential cryptanalysis in more detail.

1.2 Differential Cryptanalysis

One of the main ways to analyze the security of a block cipher construction is the field of differential cryptanalysis. It is one of the oldest cryptanalysis techniques, first proposed back in the late 1980s by Biham and Shamir [BS90], although the original designers of the Data Encryption Standard (DES) stated that they knew about the concept of differential cryptanalysis as early as 1974 and that security against it was one of the design goals [Cop94].

Differential cryptanalysis is (usually) a chosen-plaintext attack, based on pairs of plaintexts with an attacker-chosen difference in some bits. This difference

is usually added via the XOR function. By observing the propagation of these differences throughout the cipher structure, an attacker is hoping to detect a statistical pattern in the distribution of the resulting ciphertexts.

The Substitution-Box (S-Box) step of block ciphers is of special importance for differential cryptanalysis. We look at a pair of values x_1, x_2 and their difference

$$\Delta x = x_1 \oplus x_2.$$

We then compute the output of these values after the application of the S-Box layer S,

$$y_1 = S(x_1), \quad y_2 = S(x_2)$$

and their difference

$$\Delta y = y_1 \oplus y_2.$$

Due to the deterministic nature of the S-Box it follows that if $\Delta x = 0$, then Δy must also be 0. However, for non-zero differences in the input, we also have non-zero differences in the output. One common way to represent the differential behavior of an S-Box is a so-called Differential Distribution Table (DDT). It is generated by enumerating all possible combinations of input values x_1, x_2, calculating Δx and Δy, and incrementing the respective cell in the DDT.

Example. We can take the following S-Box S_1 borrowed from [KR11] and analyze its differential behavior using a DDT.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_1(x)$	6	4	c	5	0	7	2	e	1	f	3	d	8	a	9	b

This S-Box S_1 results in the DDT given in Table 1.1. We observe that the probability that an input difference of `f` maps to an output difference of `d` is especially high: $\mathrm{DDT}(\mathtt{f}, \mathtt{d}) = \frac{10}{16}$.

An attacker can now mount attacks on a cipher using this S-Box by choosing input plaintext pairs with a difference of `f`, and they will map to an output difference of `d` with a probability of $\frac{10}{16}$. By stitching together multiple of these small high-probability transitions, an attacker can get a high-probability

$\Delta x \setminus \Delta y$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	6	-	-	-	-	2	-	2	-	-	2	-	4	-
2	-	6	6	-	-	-	-	-	-	2	2	-	-	-	-	-
3	-	-	-	6	-	2	-	-	2	-	-	-	4	-	2	-
4	-	-	-	2	-	2	4	-	-	2	2	2	-	-	2	-
5	-	2	2	-	4	-	-	4	2	-	-	2	-	-	-	-
6	-	-	2	-	4	-	-	2	2	-	2	2	2	-	-	-
7	-	-	-	-	-	4	4	-	2	2	2	2	-	-	-	-
8	-	-	-	-	-	2	-	2	4	-	-	4	-	2	-	2
9	-	2	-	-	-	2	2	2	-	4	2	-	-	-	-	2
a	-	-	-	-	2	2	-	-	-	4	4	-	2	2	-	-
b	-	-	-	2	2	-	2	2	2	-	-	4	-	-	2	-
c	-	4	-	2	-	2	-	-	2	-	-	-	-	-	6	-
d	-	-	-	-	-	-	2	2	-	-	-	-	6	2	-	4
e	-	2	-	4	2	-	-	-	-	-	2	-	-	-	-	6
f	-	-	-	-	2	-	2	-	-	-	-	-	-	10	-	2

Table 1.1: Differential Distribution Table (DDT) of S-Box S_1.

differential characteristic for multiple rounds of the block cipher. The overall probability of that characteristic is the product of the individual S-Box transition probabilities. The next part of the attack involves requesting enough pairs from an encryption oracle so that enough pairs follow this differential characteristic. The attacker can then guess parts of the last round-key, and compute backwards to check the validity of the key guess. This is done by comparing the expected allowed differences and the computed difference. We can reduce the key space by filtering keys that lead to impossible transitions. Repeating this process over multiple iterations allows to filter the key space down to a single result. This approach allows the attacker to recover (parts of) the secret key.

This approach is still valid for modern block ciphers and the main factor impacting the time and data complexity of this kind of attacks is the overall probability of the best differential characteristic. Therefore, modern block ciphers follow some guidelines to decrease the theoretical maximum probability of differential characteristics:

- The maximal differential probability should be as low as possible (for a 4-bit S-Box like in the previous example, the maximum differential probability should not exceed $\frac{4}{16} = \frac{1}{4}$.)
- The linear mixing layers should ensure a large number of active S-Boxes for the cipher (An S-Box is active with regards to differential cryptanalysis if it has a non-zero difference).

A variant of differential cryptanalysis is "truncated" differential cryptanalysis. This method looks only at a reduced version of the cipher, where the n-bit S-Boxes are represented only by one bit, indicating whether the S-Box is active. This allows for a simplified view of the differential characteristic and is often used to greatly reduce the search space when using automated search tools. However, due to the simplified view, a truncated differential characteristic can lead result in contradictions when extended to a full differential characteristic.

Differential cryptanalysis is the main attack technique used in this work.

1.3 Tweakable Block Ciphers

Liskov et al. [LRW02] defined a few basic properties a tweakable block cipher must have:

- In addition to the *key* and the *message*, a tweakable block cipher has a third input called *"tweak"*. This can be seen in Figure 1.1. A tweakable block cipher is a family of permutations, and each *key-tweak* pair selects one permutation.
- The tweakable block cipher must be secure even if the attacker is able to control the *tweak* input.

It should also be able to change this *tweak* efficiently, e.g., no expensive key-schedule operations should be involved.

In addition to these properties, they also proposed two methods to construct secure tweakable block ciphers from existing block ciphers:

Given a block cipher $E_K(M)$, where K is the secret key and M is the message, we can construct a secure tweakable block cipher with an additional tweak input

(a) Block cipher.

(b) Tweakable block cipher.

Figure 1.1: Differences between classical and tweakable block ciphers.

T as follows [LRW02]:

$$E_K(M,T) = E_K(T \oplus E_K(M))$$

The second method makes use of a secure hash function h to improve overall efficiency. Using this hash function we can construct a secure tweakable block cipher as follows [LRW02]:

$$E_K(M,T) = E_K(M \oplus h(T)) \oplus h(T)$$

These constructions are proven to be secure, however they are considerably slower than the basic block cipher itself. Because of this, there has been a push to design more dedicated lightweight tweakable block ciphers which still provide strong security bounds.

Tweakable block ciphers are also a useful building block for authenticated encryption. Many of the candidates in the ongoing CAESAR authenticated encryption competition use or introduce tweakable block ciphers as an internal building block (e.g., Deoxys [Jea+16], a TWEAKEY [JNP14] design).

1.3.1 TWEAKEY

Jean et al. [JNP14] proposed the TWEAKEY framework to "unify the design of tweakable block ciphers". In contrast to the approach taken by Liskov et al. [LRW02], which uses existing block ciphers as a black-box to create a secure tweakable block cipher, their approach is based on adapting key-alternating ciphers. They introduce the concept of a *tweakey* input, which can refer to both *key* or *tweak* material. This aims to unify the importance of both the *key*

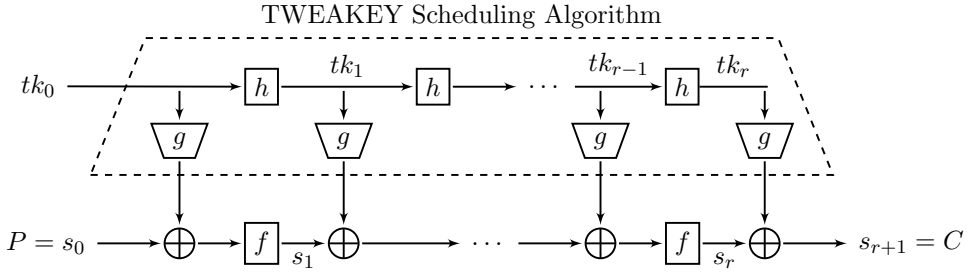


Figure 1.2: The TWEAKEY framework (see [LRW02]).

and *tweak*, with the reasoning that *key* material obviously has to kept secret, however the *tweak* input may be controllable by the attacker and is therefore also of high importance.

One of the advantages gained by this approach is that existing work on key schedule design can be leveraged to build a secure *tweakey* schedule. The TWEAKEY framework consists of three components:

- A subtweakey extraction function g, to extract a subtweakey from the tweakey state and incorporate it into the internal state (the XOR function would be a simple choice).
- An internal state update function f, which is analogous to the round function for classical block ciphers.
- A tweakey state update function h, which is analogous to the key schedule for classical block ciphers.

The complete framework with its three core components can be seen in Figure 1.2.

1.3.2 MANTIS

Beierle et al. [Bei+16] presented a family of tweakable block ciphers called SKINNY. The SKINNY family is designed to be a high-performance competitor to other modern block ciphers, while still providing strong security guarantees. In addition, they presented a dedicated variant of SKINNY called MANTIS, designed as an efficient solution for memory encryption.

MANTIS is a low-latency tweakable block cipher, which aims to be competitive with PRINCE while also adding a tweak input. Turning PRINCE into a tweakable block cipher can be done by using the TWEAKEY framework. PRINCE's round function follows the well-understood AES structure, with the overall design being symmetric around an inner linear layer, enabling the α-reflection property of PRINCE. α-reflection is a property coined by the designers of PRINCE. Through the use of a symmetric cipher structure and a smart choice of the round-constants, the structure used for encryption can also be used for decryption, decreasing the area of the cipher circuit. Using the PRINCE round function in the TWEAKEY framework in combination with a good tweak schedule creates a secure tweakable block cipher. However, to be secure against related-tweak attacks, the number of rounds needs to be increased to a certain level, in turn increasing the overall latency of the cipher.

The goal of MANTIS is to find a design using components that provide sufficient security with a minimal number of rounds. Therefore, some components of PRINCE were replaced by their counterparts from the then recently proposed low-energy block cipher MIDORI [Ban+15]. Both the S-Box as well as the ShiftRows step were taken from MIDORI, while keeping the symmetrical structure and the resulting α-reflection property, as well as the FX-construction from PRINCE. The designers of MANTIS also presented a tweak schedule that ensures a high number of active S-Boxes even if the attacker can control the tweak input.

A detailed description of the tweakable block cipher MANTIS can be found in the design paper [Bei+16], and short summary description will be given in section 2.1.

1.3.3 QARMA

QARMA is a tweakable block cipher designed by Avanzi [Ava16]. Its core design is very similar to MANTIS, with a few changes made to improve resistance against certain attacks. Furthermore, QARMA will be used in ARMv8 [Bra16] for pointer authentication.

The first change concerns the MixColumns step and is one of the main contributions of his paper: A family of Almost MDS matrices that allows to encode rotations, while allowing to keep the minimal latency of $\{0, 1\}$-matrices. The

second change is concerning the S-Box used in MANTIS. Avanzi describes search heuristics for low-latency S-Boxes and gives a choice of three distinct S-Boxes that can be used in QARMA. The final changes is to the inner round of the cipher. QARMA is a three-round Even-Mansour scheme that uses a keyed permutation in the inner round. Finally, the tweak schedule is updated to incorporate a linear feedback shift register for some of the state cells.

A detailed description of the tweakable block cipher QARMA can be found in the design paper [Ava16], and a short summary description will be given in section 2.2.

1.4 Summary of Contributions of this Thesis

First, we provide a practical implementation to an existing attack on MANTIS (in particular the MANTIS_5 variant) by Dobraunig et al. [Dob+16]. We then extend the existing methods for this kind of attack and use these improvements to attack the 6-round variant of MANTIS. We present a key-recovery attack on MANTIS_6, which can recover the 128-bit secret key using $2^{56.72}$ chosen plaintexts and $2^{56.72}$ time complexity. Finally, we try to apply the attack concept to other modern tweakable block ciphers and show how the different design of these other ciphers influence the result.

1.5 Overview

In chapter 1, we gave a short overview and basic summary of the topics relevant to this thesis. Chapter 2 contains a description of the ciphers which are attacked in this thesis. Their structure and components are listed in detail and differences are highlighted. Following that, in chapter 3 we give a short summary of an existing attack on the cipher MANTIS and present a practical implementation of the attack. We also give additional insights that were discovered during the practical implementation. In chapter 4, we improve the methods used to search for families of differential characteristics. We present a semi-automated toolchain that searches for truncated differential characteristics, extends them to full differential characteristics, combines them to a family of differential

characteristics and finally calculates the overall probability of the family. This toolchain is then used to improve the attack described in chapter 3. Using these new methods, we present a new attack on the 6-round variant of MANTIS in chapter 5. We describe this key-recovery attack on MANTIS_6 in detail and give additional comments on search strategies and the security of the 7-round variant, MANTIS_7. In chapter 6, we try to apply the idea of the attack to other tweakable block ciphers. The methods from chapter 4 are modified and results and encountered difficulties are discussed. Finally, we give a summary and conclusion in chapter 7.

2 Description of Target Ciphers

2.1 MANTIS

MANTIS is a tweakable block cipher published at CRYPTO 2016 by Beierle et al. [Bei+16]. MANTIS is the dedicated variant of their SKINNY block cipher family designed for low-latency applications, such as memory encryption. To achieve this low-latency design, they use the α-reflection property of PRINCE by Borghoff et al. [Bor+12], while using parts of the round function of MIDORI by Banik et al. [Ban+15]. The tweak is added using a variant of the TWEAKEY framework by Jean et al. [JNP14].

The designers proposed multiple variants of MANTIS that differ only in the number of rounds. MANTIS_5 is a 5-round variant, with a security claim that no related-tweak attack is possible using 2^d less than 2^{30} chosen or 2^{40} known plaintexts and 2^{126-d} time. MANTIS_7 is the full 7-round variant with a security claim that no attacks are possible using 2^n chosen plaintext/ciphertext pairs and 2^{126-n} encryption function calls.

MANTIS takes 64-bit plaintext blocks $M = M_0||M_1||\dots||M_{15}$, a 64-bit tweak $T = T_0||T_1||\dots||T_{15}$ and a $(64+64) = 128$-bit key $K = (k_0 + k_1)$. The internal state S is composed of 4×4 state cells S_i, which are 4-bit in size:

$$S = \begin{array}{|c|c|c|c|} \hline S_0 & S_1 & S_2 & S_3 \\ \hline S_4 & S_5 & S_6 & S_7 \\ \hline S_8 & S_9 & S_{10} & S_{11} \\ \hline S_{12} & S_{13} & S_{14} & S_{15} \\ \hline \end{array} .$$

The cipher's internal structure is composed of r forward rounds $\mathcal{R}$ and r backwards rounds $\mathcal{R}^{-1}$. They are connected with an involutory, unkeyed middle

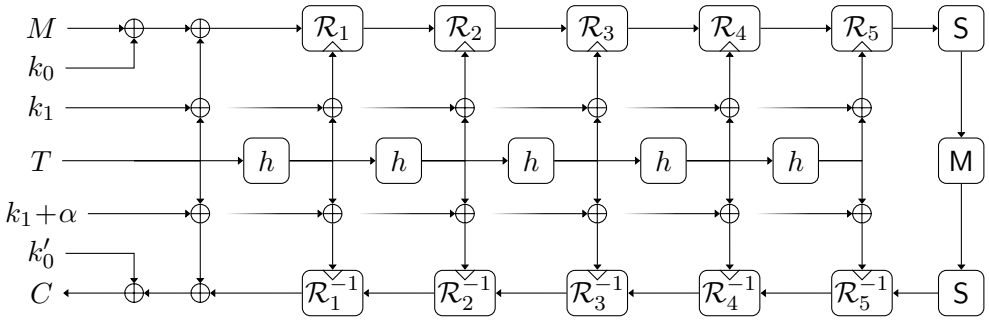


Figure 2.1: PRINCE-like structure of MANTIS$_r$, illustrated for MANTIS$_5$.

layer composed of SubCells ∘ MixColumns ∘ SubCells. The two key parts k_0 and k_1 serve different purposes. The subkey k_0 and its derived counterpart $k_0' = (k_0 \ggg 1) + (k_0 \gg 63)$ are used as whitening keys. The subkey k_1 is used as the round key for the round functions $\mathcal{R}$ and $\mathcal{R}^{-1}$, and is added in conjunction with the tweak T according to the TWEAKEY construction. The overall structure of MANTIS is illustrated in Figure 2.1.

Round Functions $\mathcal{R}$ and $\mathcal{R}^{-1}$. The round functions are based on the round function of MIDORI. The internal state is updated using a sequence of state transformations:

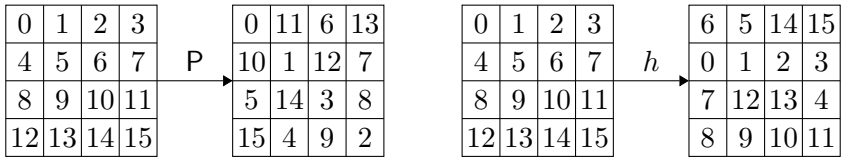
$$\mathcal{R} = \mathsf{MixColumns} \circ \mathsf{PermuteCells} \circ \mathsf{AddTweakey} \circ \mathsf{AddConstant} \circ \mathsf{SubCells}.$$
$$\mathcal{R}^{-1} = \mathsf{SubCells} \circ \mathsf{AddConstant} \circ \mathsf{AddTweakey} \circ \mathsf{PermuteCells}^{-1} \circ \mathsf{MixColumns}.$$

SubCells (S). The used S-Box $\mathcal{S}$ is taken from MIDORI [Ban+15]. It is involutory and takes a 4-bit input and has a 4-bit output:

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

AddTweakey (A) and AddConstant (C). For $\mathcal{R}$, the round constant C_i, the subkey k_1 and the round tweakey $T_i = h^i(T)$ are added to the state using XOR. For $\mathcal{R}^{-1}$, the constant α is added in addition to the previously mentioned values. The round tweakey is updated via the tweakey update function h, which is given in Figure 2.2b.

PermuteCells (P). The 4×4 cells of the state are permuted by P, given in Figure 2.2a.



(a) State permutation P. (b) Tweakey update permutation h.

Figure 2.2: The permutations P and h used in MANTIS.

MixColumns (M). The state is multiplied column-wise with the following matrix M over GF(2^4). M is an involutory, near-MDS matrix:

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

2.2 QARMA

QARMA is a tweakable block cipher designed by Avanzi [Ava16]. QARMA's design and overall structure is very similar to that of MANTIS, but it aims to improve a few of the design choices made by the authors of MANTIS to improve the overall security. The designer proposed several components that can be used in the round function of QARMA, leaving the final choice to the user. Additionally, QARMA offers two variants for its state size, a 64-bit and a 128-bit variant. We will only summarize the QARMA-64 variant of the cipher

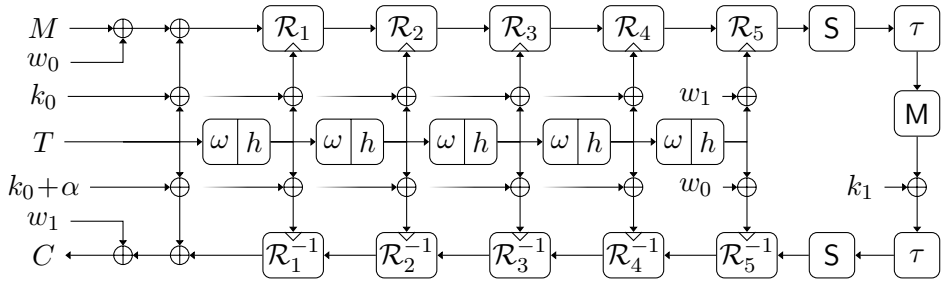


Figure 2.3: Structure of QARMA_r, illustrated for QARMA_5.

in the following and refer to the design paper [Ava16] for more details on the 128-bit variant.

The internal state is mapped to 16 4-bit state cells, arranged as a 4×4 matrix S:

$$S = \begin{array}{|c|c|c|c|} \hline S_0 & S_1 & S_2 & S_3 \\ \hline S_4 & S_5 & S_6 & S_7 \\ \hline S_8 & S_9 & S_{10} & S_{11} \\ \hline S_{12} & S_{13} & S_{14} & S_{15} \\ \hline \end{array} .$$

The plaintext is given as $M = M_1||M_2||\ldots||M_{15}$ and the tweak is given as $T = T_0||T_1||\ldots||T_{15}$, so they can be easily mapped to the internal state representation.

QARMA is a three-round Even-Mansour construction and its overall structure can be seen in Figure 2.3.

SubCells (S). The designer proposed three different S-Boxes that can be used with QARMA.

σ_0 is a lightweight, involutory S-Box with only two fixed points (in comparison to the 4 fixed points of the MIDORI S-Box used in MANTIS). Additionally it has the same depth (3.5) as the MIDORI S-Box and an area of 14 GE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma_0(x)$	0	e	2	a	9	f	8	b	6	4	3	7	d	c	1	5

σ_1 is a homogeneous, involutory S-Box. It has a depth of 4 and an area of 16 GE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma_1(x)$	a	d	e	6	f	7	3	5	9	8	0	c	b	1	2	4

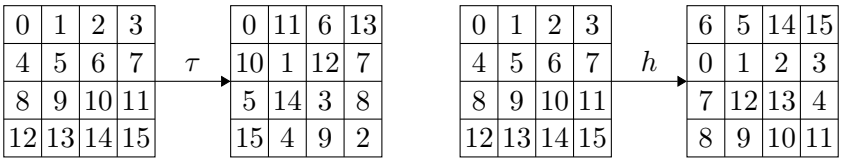
σ_2 is an affine equivalent of the S-Box S_6 from the PRINCE family. σ_2 and its inverse have a depth of 4.5 and 4 respectively and and area of approximately 20 GE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\sigma_2(x)$	b	6	8	f	c	0	9	e	3	7	4	5	d	2	1	a

These three S-Boxes have been carefully selected by the designer of QARMA, following a few desirable cryptographic properties. Furthermore, the user has the final choice over the used S-Box and can consider their cryptographic properties and latency. We refer to the design document [Ava16] for a more detailed description and the rationale behind the choice of these S-Boxes.

AddTweakey (A) and AddConstant (C). For $\mathcal{R}$, the round constant C_i, the subkey k_1 and the round tweakey $T_i = h^i(T)$ are added to the state using XOR. For $\mathcal{R}^{-1}$ the constant α is added in addition to the previously mentioned values. The round tweakey is then updated via the tweakey update function: The tweakey permutation h (given in Figure 2.4b) is applied to the tweak, and in addition a linear feedback shift register (LFSR) ω is applied to the tweak state cells $T_0, T_1, T_3, T_4, T_8, T_{11}$ and T_{13}. ω is a maximal period LFSR that maps a cell (b_3, b_2, b_1, b_0) to $(b_0 + b_1, b_3, b_2, b_1)$.

PermuteCells (P). The 4×4 cells of the state are permuted by τ, given in Figure 2.4a. Note that QARMA, like MANTIS, also reuses the MIDORI cell permutation, therefore $\tau = \mathsf{P}$.



(a) State permutation τ. (b) Tweakey update permutation h.

Figure 2.4: The permutations τ and h used in QARMA.

MixColumns (M). The state is multiplied column-wise with the following matrix $M_{4,3}$ over GF(2^4). $M_{4,3}$ is an involutory, Almost-MDS matrix:

$$M_{4,3} = \begin{pmatrix} 0 & \rho & \rho^2 & \rho \\ \rho & 0 & \rho & \rho^2 \\ \rho^2 & \rho & 0 & \rho \\ \rho & \rho^2 & \rho & 0 \end{pmatrix}.$$

ρ denotes a circular rotation of the bits by one (to the left), i.e., (b_3, b_2, b_1, b_0) to (b_2, b_1, b_0, b_3).

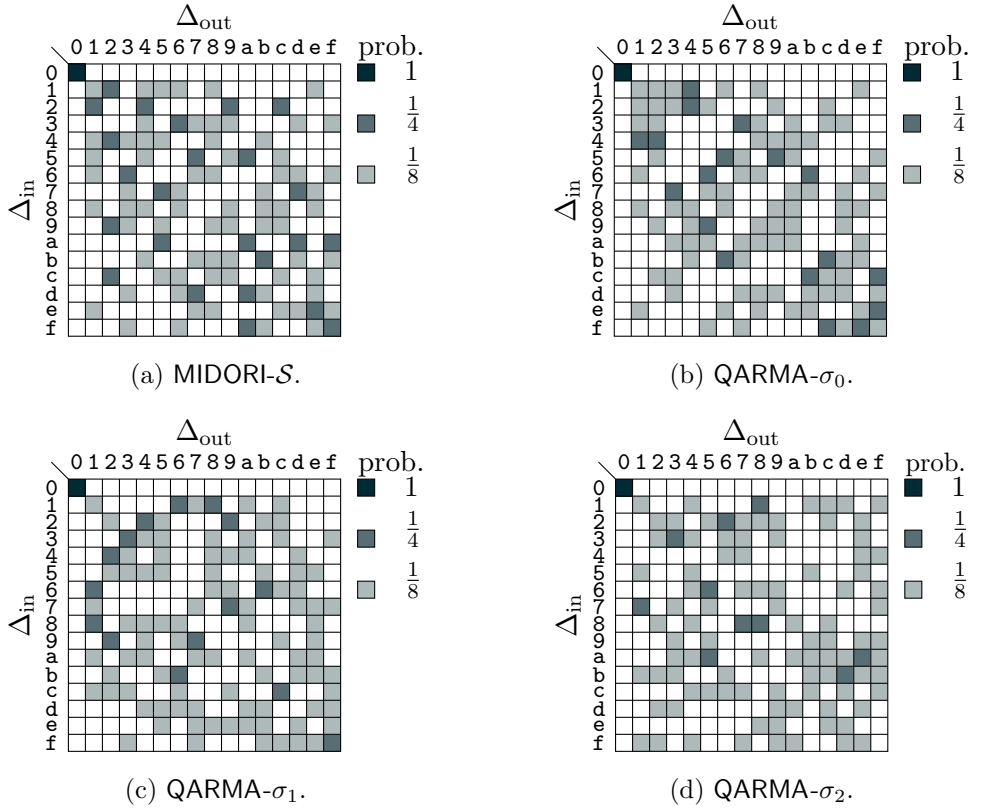
2.3 Differences between MANTIS and QARMA

QARMA-64's main differences to MANTIS are:

- A choice of three different S-Boxes $\sigma_0, \sigma_1, \sigma_2$.
- A new MixColumns layer using the matrix $M_{4,1}$.
- An improved, keyed inner round.
- The addition of a LFSR to the tweak update function.

We will now go into more detail about each of these properties.

The S-Boxes $\sigma_0, \sigma_1, \sigma_2$. The DDTs of the MIDORI S-Box used in MANTIS and the three S-Boxes $\sigma_0, \sigma_1, \sigma_2$ used in QARMA can be seen in Figure 2.5. We can see the different number of high-probability differential transitions for the four S-Boxes. Avanzi suggests that the S-Box σ_0 could be used as a replacement for the MIDORI S-Box used in MANTIS, due to their similar area and behavior of their fixed points. The differential behavior of QARMA's S-Boxes is not as



(a) MIDORI-$\mathcal{S}$. (b) QARMA-σ_0.

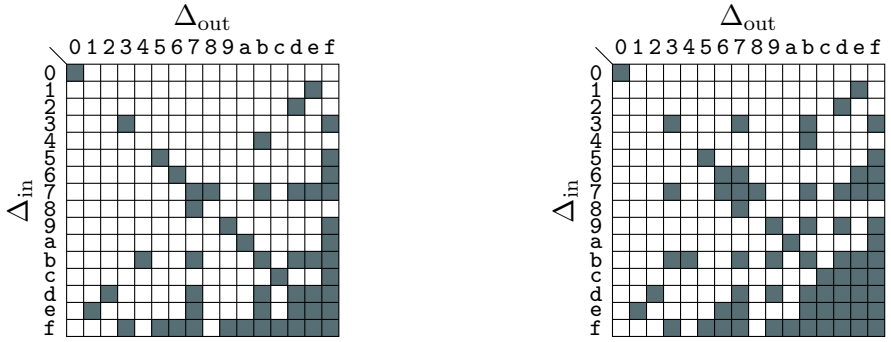
(c) QARMA-σ_1. (d) QARMA-σ_2.

Figure 2.5: Differential Distribution Tables (DDT) of the S-Boxes used in MANTIS and QARMA.

beneficial for the attacks described in this thesis as the differential behavior of the MIDORI S-Box.

The new linear layer $M_{4,1}$. The matrix M used in MANTIS has the property that all transitions with a branch number of 4 are valid if all 4 active cells have the same difference. This is not the case with the matrix $M_{4,1}$ used in QARMA, since the combination of different rotations prevent this fact. This makes it harder to propagate chosen differences through the cipher structure. The linear layer $M_{4,1}$ is designed with the observation that it is important "that the diffusion layer does not always map the bits to the same places and the

round constants also influence other bits, to prevent characteristics that map a small subset of possible states onto itself." [Ava16]. This is one of the properties that we use in the attacks described in this thesis, and this new linear layer has a large impact on the probability of families of differential characteristics.



(a) M used in MANTIS. (b) $M_{4,3}$ used in QARMA.

Figure 2.6: DDTs for the truncated differential behavior of the different MixColumns operations.

The inner round construction. The inner round function of QARMA has two major differences to the inner round function of MANTIS. First, an additional PermuteCells step is added in both the forward and backwards direction. Second, a key addition is added in between the two SubCells steps. This removes one of the negative properties of MANTIS. MANTIS's inner round is essentially the steps SubCells–MixColumns–SubCells. This creates a super-box property, meaning the two SubCells steps and the MixColumns step can be combined to one large super step, which has improved probabilities for some transitions. This property is used in the attacks described in this thesis.

The tweak update function. The tweak update function of QARMA includes an LFSR in addition to the tweak state permutation h (which is also used in MANTIS). This makes it harder for an attacker to construct related-tweak attacks, since it is not possible to keep differences in the tweak state constant. Since this LFSR does not exist in MANTIS, it is possible to keep the differences in the tweak constant. This property is used in the attacks described in this thesis.

3 Verification of Key-Recovery Attack on MANTIS$_5$

Dobraunig et al. [Dob+16] proposed a key-recovery attack on MANTIS$_5$ which violates the security claims given by the designers. We showed that the attack is practical and verified its theoretical time and data complexities by providing a practical implementation and discovered additional insights. This implementation and the insights discovered directly contributed to the paper by Dobraunig et al. [Dob+16].

This chapter is structured as follows. First the attack by Dobraunig et al. and its structure are summarized in section 3.1. Section 3.2 is dedicated to the practical implementation and its details.

3.1 Summary of Key Recovery Attack on MANTIS$_5$

The designers of MANTIS analyzed their proposal with respect to differential attacks using a mixed-integer linear program (MILP) that models the differential behavior of the cipher (truncated to state cells). Using this truncated model, they analyzed the lower bound of active S-Boxes and computed the minimum number of active S-Boxes to be 34 for the 5-round version of MANTIS. The best differential probability of the S-Box is 2^{-2}, resulting in an overall upper limit of $2^{-2 \cdot 34} = 2^{-68} < 2^{-64}$ for the probability of a differential characteristic. Therefore, the designers concluded that "no related tweak linear or differential distinguisher based on a characteristics is possible for MANTIS$_5$" [Bei+16] and that MANTIS$_5$ is secure against "practical" attacks with a data complexity 2^d at most 2^{30} chosen plaintexts and computational complexity of 2^{126-d}.

The attack by Dobraunig et al. is based on a differential characteristic with 36 active S-Boxes. This differential characteristic was found using the MILP model

of the truncated version of MANTIS$_5$. Although it does not match the lower bound of 34 S-Boxes, its properties are superior for the attack when compared to solutions with 34 active S-Boxes.

Consider the properties of the used S-Box, in particular the differential distribution table given in Figure 2.5a. All transitions from and to `a` have the highest differential probability of $\frac{1}{4}$. Furthermore, the transition `a` $\rightarrow$ `a` is included. Since MixColumns has only binary coefficients, all transitions that match its branch number of 4 are valid when all active cells have a difference of `a`.

This means that if a truncated differential characteristic exists where the MixColumns step has only transitions with a branch number of 4, all differences in the truncated characteristic can be set to `a` and a valid differential characteristic with the highest possible probability is obtained.

The structure of the S-Box allows for more improvement. By allowing more differences in some state cells instead of just `a`, essentially clustering multiple differential characteristics together, the overall probability can be improved further. Due to the differential properties of the S-Box, values in the set that map to and from `a`, namely $\{\texttt{5}, \texttt{a}, \texttt{d}, \texttt{f}\}$, are of particular interest. By carefully choosing the allowed differences in the state cells and using the last round for key recovery, the overall probability of the characteristic can be improved from 2^{72} to $2^{40.51}$. This family of differential characteristics can be seen in Figure 3.1.

Generating and pre-filtering plaintext-ciphertext pairs. This improved probability is still not enough to perform an attack that violates the security claims given by the designers of MANTIS. The data complexity of such an attack should not exceed 2^{30} chosen plaintexts. By using a smart structure of initial plaintexts and combining them, the overall data complexity can be reduced greatly. Using the cells of the plaintext with an allowed set of differences of $\{\texttt{5}, \texttt{a}, \texttt{d}, \texttt{f}\}$, we can generate the necessary 2^{41} plaintext pairs using only 2^{28} chosen plaintext-tweak queries (this process is described in more detail in section 5.2).

Once enough plaintext-tweak-ciphertext pairs have been generated, they can be pre-filtered based on their ciphertext differences. Cells $S_1, S_4, S_{11}, S_{13}, S_{15}$ have a difference of zero. Random plaintext pairs fulfill this property with a probability of 2^{-24}, so the total amount of 2^{41} plaintext-tweak-ciphertext pairs can be filtered down to 2^{19} pairs $i \in I_r$.

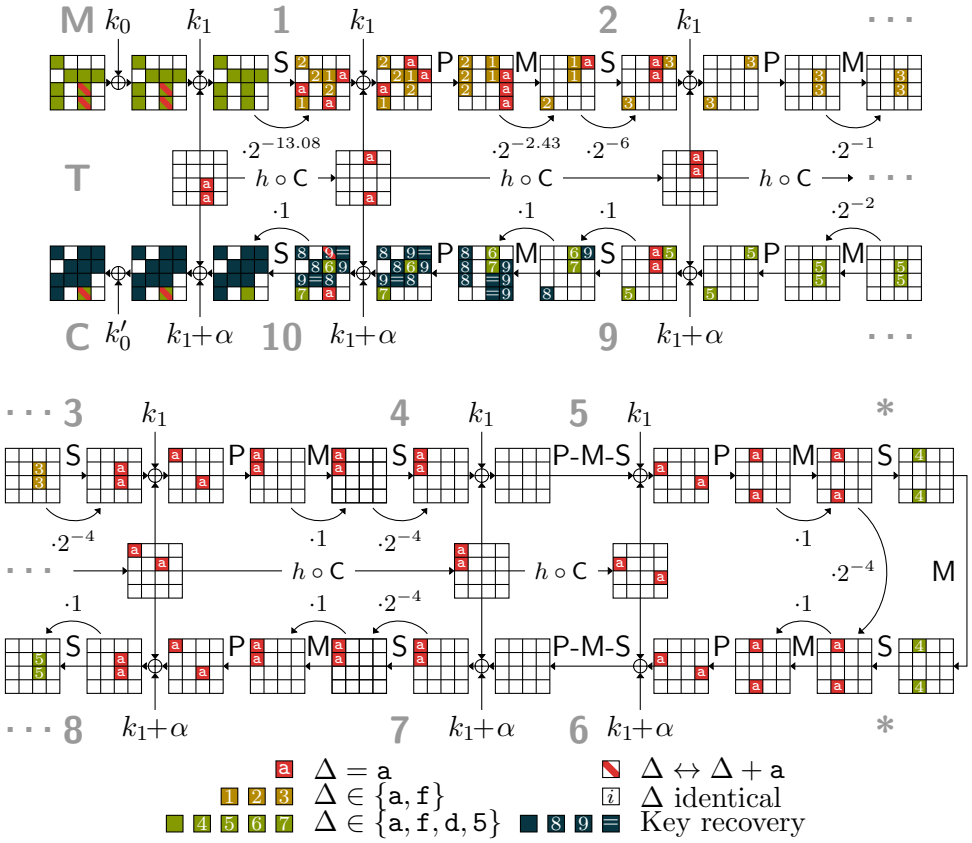


Figure 3.1: Family of differential characteristics for MANTIS$_5$ (see [Dob+16]).

Recovering 44 bits of k'_0+k_1. After filtering the generated pairs, the next step is the recovery of 44 bits of the final whitening key $k'_0 + k_1$. We guess parts of the key corresponding to the columns of the MixColumns step in round 10. The overall probability that a 44-bit keyguess follows the family of characteristics in Figure 3.1 up to that point is 2^{-30}. This means for each pair, $2^{44-30} = 2^{14}$ key guesses remain which satisfy the conditions. For the total 2^{19} pairs, this results in $2^{19} \cdot 2^{14} = 2^{33}$ keyguesses remaining, reducing the keyspace by a factor of 2^{11}. To filter the keyspace down to a single correct key, this process needs to be repeated a total of 4 times. For each repetition $r \in 1, 2, 3, 4$, we get a set of 2^{33} possible keys and their intersection results in the single correct 44-bit subkey.

Recovering 32 bits of $k_0 + k_1$. Using the recovered 44 bits of key information, the plaintext-tweak-ciphertext pairs can be filtered down to just the pairs following the family of characteristics. A wrong pair gets identified as a correct one with a probability of 2^{-30}, so only the correct 4 pairs should remain. Using these correct pairs, we can compute forward through the S-Box layer of round 1, and check our keyguess for cells $S_0, S_5, S_6, S_7, S_8, S_{10}, S_{12}, S_{14}$. The overall probability that a wrong key matches the family of characteristics for all correct pairs is $2^{-62.04}$. Therefore, only the correct 32-bit subkey should remain.

Recovering the full key $k_0 + k_1$. Up until now 76 bits of the key material have been recovered. Using the structure of rounds 2, 3, 8 and 9, we can recover 14 more keybits using a guess-and-determine approach. This leaves us with 38 bits of the full key that are unknown. To recover the full secret key, we guess the remaining 38 key bits and perform 2^{38} trial encryptions.

3.2 Practical Implementation of Key Recovery Attack on MANTIS$_5$

Since the attack described in section 3.1 has a complexity requirement that is close to "practical", one of the goals of this thesis was to provide an implementation of the attack. Especially for modern ciphers, attacks that are able to be executed on a standard single-core CPU in a reasonable time frame (less than one day) are quite rare. An implementation of the attack serves to both validate the correctness of the theoretical attack as well as the proposed time and data complexity. Furthermore, some minor issues with the theoretical attack arose during the practical implementation and corresponding workarounds were found and implemented. The structure of the implementation with all issues, workarounds and optimizations is described in the following.

Generating and pre-filtering plaintext-ciphertext pairs. The complexity of the trivial approach to combine the chosen plaintext-tweak-ciphertext pairs is $4 \cdot 2^{41}$ state XOR operations. This would dominate the time complexity of the attack. However, a more efficient solution exists. Instead of iterating all possible plaintext-tweak-ciphertext pair combinations and then filtering based on the

difference in the ciphertext, we can group each single plaintext-tweak-ciphertext into partitions based on the values of the ciphertext cells $S_1, S_4, S_{11}, S_{13}, S_{15}$. Since these cells have a difference of zero in our characteristic, we can then only iterate over all possible combinations in each partition. Since we query 2^{25} plaintexts for both the tweak and the tweak with differences, the expected size of each partition is 2^5. Combining each of those 2^5 candidates with their respective 2^5 possible matches for all 2^{20} partitions brings the total complexity down to $2^5 \cdot 2^5 \cdot 2^{20} = 2^{30}$ state XOR operations. This results in about 2^{19} plaintext-tweak-ciphertext pairs that follow the constraints in the plaintext, the tweak and the ciphertext states. This part takes about 16 minutes on a single desktop core.

Recovering 44 bits of $k'_0 + k_1$. We now need to guess 44 bits of the subkey $k'_0 + k_1$ and verify the keyguess against the $4 \cdot 2^{19}$ pairs remaining after the pre-filtering phase. To verify a keyguess for one pair, we need to perform $2 \cdot 11$ S-Box lookups. Using a trivial approach, this would result in $2 \cdot 11 \cdot 2^{44+19+2} \approx 2^{69.46}$ S-Box lookups in total, which corresponds to roughly $2^{61.87}$ MANTIS$_5$ encryptions (based on the total number of $12 \cdot 16$ S-Boxes in MANTIS$_5$). However, we do not need to guess the whole 44 bit subkey at once. The 4 conditions described in section section 3.1 are independent of each other, meaning we can guess (and verify) a 16-bit subkey, two 12-bit subkeys and a 4-bit subkey separately from each other. This reduces the total number of S-Box lookups to $2 \cdot (2^{16} \cdot 4 + 2 \cdot 2^{12} \cdot 3 + 2^4) \approx 2^{19.13}$ or an equivalent of $2^{11.54}$ MANTIS$_5$ encryptions per pair. For the total number of $4 \cdot 2^{19}$ pairs that are left after the pre-filtering phase, this results in a time complexity corresponding to approximately $2^{32.54}$ MANTIS$_5$ encryptions, which is a huge improvement to the trivial version.

This phase of the key recovery is dominating the data complexity requirement. When performing this step in a straightforward implementation, we get 4 lists of 2^{33} key candidates. We then need to find intersections between the lists which adds a computational complexity of about 2^{33}. However, it is again not necessary to store the full 44-bit key guess. Since the 4 parts of the key are independent, we can store them in 4 separate sets. Set $C_{14}^{r,i}$ holds the valid 4-bit key guesses for the conditions in column 2 of the MixColumns step in round 10, set $C_{0,5,10}^{r,i}$ holds the valid 12-bit keyguesses for the conditions in column 1, set $C_{2,7,8}^{r,i}$ holds the valid 12-bit keyguesses for the conditions in column 4

and set $C^{r,i}_{3,6,9,12}$ holds the valid 16-bit keyguesses for the conditions in column 3 for a given repetition $r \in 1, 2, 3, 4$ and pair $i \in I_r$. The expected set of 2^{14} key candidates is the product of these 4 sub-sets.

We refer to this structured set of key candidates as a bundle $\mathcal{B}^{(r,i)}$, where

$$\mathcal{B}^{(r,i)} = \mathcal{C}^{(r,i)}_{0,5,10} \times \mathcal{C}^{(r,i)}_{14} \times \mathcal{C}^{(r,i)}_{3,6,9,12} \times \mathcal{C}^{(r,i)}_{2,7,8}.$$

Storing all bundles requires only about $4 \cdot 2^{19} \cdot 10.25 < 2^{25}$ MANTIS states. To find the correct value of all 44 bits, we now need to compute

$$\bigcap_{r=1}^{4} \bigcup_{\substack{i \in I_r \\ |I_r| \approx 2^{19}}} \mathcal{C}^{(r,i)}_{0,5,10} \times \mathcal{C}^{(r,i)}_{14} \times \mathcal{C}^{(r,i)}_{3,6,9,12} \times \mathcal{C}^{(r,i)}_{2,7,8}.$$

The computational complexity of intersecting these lists is again lowered when starting the intersection with the most restrictive subkey $C^{r,i}_{3,6,9,12}$. For each possible 16-bit subkey $C^{r,i}_{3,6,9,12}$, we build a list of references to each bundle containing this subkey. Since the expected size of each set is 2^4 and we have 2^{19} bundles per iteration, the size of each list should be $\approx 2^7$. We use these lists to intersect the bundles of two repetitions as follows. We look up the containing bundles for each possible 16-bit subkey $C^{r,i}_{3,6,9,12}$ in both repetitions and intersect the keyguesses in the expanded set $\mathcal{C}^{(r,i)}_{0,5,10} \times \mathcal{C}^{(r,i)}_{14} \times \mathcal{C}^{(r,i)}_{2,7,8}$ (this expanded set should contain $\approx 2^{10}$ keyguesses). This results in an overall computational complexity of $\approx 2^{16} \cdot 2^7 \cdot 2^7 \cdot 2^{10} = 2^{40}$. Although the set intersection of the expanded set containing the $\approx 2^{33}$ key candidates would be more computationally efficient, we would need to store $2^{33} \cdot 6 \cdot 2 \approx 2^{36.58}$ bytes of key data, which is more prohibitive for a practical implementation than the additional computational complexity.

During the practical implementation, we found that for the valid 44-bit subkey, there exists one differentially equivalent subkey which also fulfills all conditions. This always occurs due to the second-order differential properties of the MIDORI S-Box (An in-depth analysis of the second-order differential properties of the S-Boxes used in MANTIS and QARMA can be found in [Eic17]). Therefore, the total number of recovered bits is only 43, but this number is still sufficient to filter the plaintext-tweak-ciphertext pairs for the next step.

Recovering 32 bits of $k_0 + k_1$. Using the recovered 43-bit subkey, the $4 \cdot 2^{19}$ plaintext-tweak-ciphertext pairs are filtered, so that only pairs that follow the family of characteristics in Figure 3.1 remain. In theory, this should result in about 4 pairs remaining, but over several runs, the average was found to be about 8 pairs (all of which follow the family of characteristics) instead. We need to perform a 32-bit key guess for each of the remaining 4 pairs and verify it by performing $2 \cdot 8$ S-Box lookups per pair and verifying cells $S_0, S_5, S_6, S_7, S_8, S_{10}, S_{12}, S_{14}$ against the family of characteristics in Figure 3.1. This results in a total of $2 \cdot 4 \cdot 8 \cdot 2^{32} = 2^{38}$ S-Box lookups, corresponding to roughly $2^{30.42}$ MANTIS$_5$ encryptions. This key can again be split in several smaller subkeys and verified independently, quite similar to the approach taken in the 44-bit key guessing phase. Due to the already acceptable computational complexity, this was not done in practice.

Again, the practical implementation revealed some difficulties.Instead of just one of the 2^{32} subkeys surviving the second phase, 2^8 subkeys stayed valid. This is again happening due to the second-order differential properties of the MIDORI S-Box. Because of this, only 24 bits of the subkey are recovered in practice.

Recovery of the full k_0 and k_1. The procedure presented in the theoretical attack first recovers 14 more bits from the second and third round, before performing 2^{38} trial encryptions to recover the final 38 bit of the secret key. This approach was not taken in the practical implementation. Instead, the full structure of the cipher was encoded as a boolean satisfiability problem [Dob+16], and the recovered $43 + 24$ bits of the subkey, as well as the filtered plaintext-tweak-ciphertext pairs are added as constraints. To easily describe the cipher structure and its bit-vector operations, the STP constraint solver [GD07] and its higher-level python bindings were used. A multithreaded SAT-solver can find the remaining bits of the secret key in less than 2 minutes on a modern desktop machine.

4 Improving Search Methods to Find a Family of Characteristics

In this chapter we will cover the basic methodology used to find families of characteristics. The different steps are explained in detail and improvements to existing methods are discussed and implemented to form a semi-automated toolchain.

4.1 MILP Model for Truncated Differential Characteristics

The truncated differential behavior of the cipher is modeled as a mixed-integer linear program (MILP). A mixed-integer linear program is an optimization program in which some of the variables are constrained to integers. The model is composed of a objective function to be minimized or maximized and several linear equalities and inequalities over $\mathbb{R}$. The model is then given to a MILP solver which minimizes the objective function and returns an assignment for all variables corresponding to the optimal solution found.

To model the truncated differential behavior of our target cipher MANTIS, we use the MILP model proposed by Eichlseder [Eic17] as a starting point. In the following we will shortly summarize the decision variables used in the MILP model and refer to the full model and its description in [Eic17].

The MILP model by Eichlseder uses the following decision variables for r-round MANTIS_r, indexed by their round $i \in \mathcal{R} = \{1, \ldots, r\}$, forward or backward direction $\pm \in \{+, -\}$, where $i^{\pm} \in \mathcal{R}^{\pm}$ is shorthand for $(i, \pm) \in \mathcal{R} \times \{+, -\}$, and cell position $b \in \mathcal{B} = \{0, \ldots, 15\}$, or state row $x \in \mathcal{X} = \{0, \ldots, 3\}$ and column $y \in \mathcal{Y} = \{0, \ldots, 3\}$, such that $b = 4x + y$ (Figure 4.1):

- $\alpha_i^{\pm}[b] \in \{0,1\}$ for $i^{\pm} \in (\mathcal{R} \cup \{0\})^{\pm}$, $b \in \mathcal{B}$: Truncated difference of cell S_b of the input and output state of SubCells in forward round $\mathcal{R}_i$ or backward round $\mathcal{R}_i^{-1}$.
- $\beta_i^{\pm}[b] \in \{0,1\}$ for $i^{\pm} \in \mathcal{R}^{\pm}$, $b \in \mathcal{B}$: Truncated difference of cell S_b of the output state of $\mathsf{AddTweakey}_i$ in forward round $\mathcal{R}_i$ (or the input state in backward round $\mathcal{R}_i^{-1}$).
- $\tau[b] \in \{0,1\}$ for $b \in \mathcal{B}$: Truncated difference of cell S_b of the tweak T. Note that we do not need to model the differences in the plaintext m and in the ciphertext c to optimize the number of active S-boxes.

The MILP model must be linear over $\mathbb{R}$ ($+$ denotes integer addition in the MILP inequalities), whereas the linear layers (MixColumns and AddTweakey) are linear over $\mathbb{F}_2$ ($+$ or $\oplus$ denotes xor). Therefore, the AddTweakey step is modeled using its differential branch number of 2. To model MixColumns, we use the description proposed in [Eic17], which uses a number of helper variables to model the exact truncated behavior of the MixColumns step used in MANTIS. The full MILP model, including all constraints and the objective function, is given in Figure 4.2. This model is later tweaked further to influence the structure of the solutions found. We list these additional constraints in section 5.7.

4.2 Differential Search Tool

The MILP solver outputs truncated differential characteristics when given the task to minimize the given MILP model. However, the truncated differential

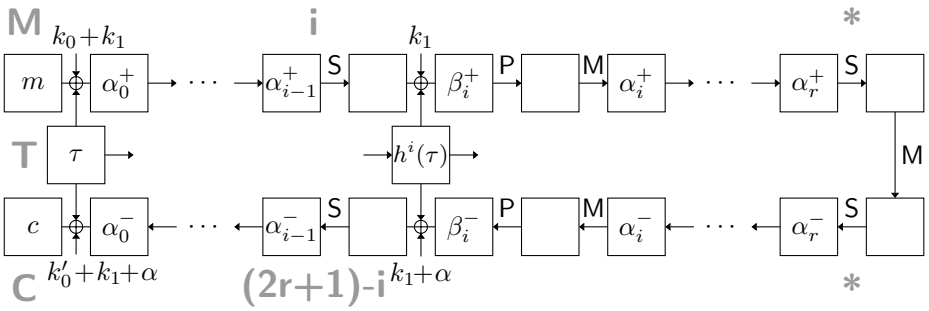


Figure 4.1: MILP variables for truncated differential model of MANTIS_r (from [Eic17]).

$$\min \sum_{i^{\pm}\in(\mathcal{R}\cup\{0\})^{\pm}} \sum_{b\in\mathcal{B}} \alpha_i^{\pm}[b] \qquad \text{(active S-boxes)}$$

$$\text{s.t.} \quad \forall b \in \mathcal{B}, i^{\pm} \in \mathcal{R}^{\pm}: \qquad (\mathsf{AddTweakey}_i)$$

$$2\chi_i^{\pm}[b] \le \alpha_{i-1}^{\pm}[b] + h^i(\tau)[b] + \beta_i^{\pm}[b] \le 3\chi_i^{\pm}[b]$$

$$\forall y \in \mathcal{Y}, i^{\pm} \in \mathcal{R}^{\pm}: \qquad (\mathsf{PermuteCells}_i,\ \mathsf{MixColumns}_i)$$

$$4\mu_{i,y}^{\pm} \le \sum_{x\in\mathcal{X}} \left(\mathsf{P}(\beta_i^{\pm})[4x+y] + \alpha_i^{\pm}[4x+y]\right) \le 8\mu_{i,y}^{\pm}$$

$$\mu_{i,y}^{\pm} \le \sum_{x\in\mathcal{X}} \mathsf{P}(\beta_i^{\pm})[4x+y], \qquad \mu_{i,y}^{\pm} \le \sum_{x\in\mathcal{X}} \alpha_i^{\pm}[4x+y]$$

$$4 \le 4\nu_{i,y,8}^{\pm} + \sum_{x\in\mathcal{X}} \nu_{i,y,x}^{\pm}$$

$$6\nu_{i,y,8}^{\pm} \le \sum_{x\in\mathcal{X}} \left(\mathsf{P}(\beta_i^{\pm})[4x+y] + \alpha_i^{\pm}[4x+y]\right)$$

$$2\nu_{i,y,8}^{\pm} = \sum_{x\in\mathcal{X}} \mathsf{P}(\beta_i^{\pm})[4x+y] - 2\nu_{i,y,9}^{\pm} - \nu_{i,y,10}^{\pm}$$

$$\nu_{i,y,10}^{\pm} = 2\nu_{i,y,x+4}^{\pm} + 1 - \nu_{i,y,x}^{\pm} - \mathsf{P}(\beta_i^{\pm})[4x+y] - \alpha_i^{\pm}[4x+y] \qquad \forall x \in \mathcal{X}$$

$$\forall y \in \mathcal{Y}: \qquad \text{(Inner MixColumns)}$$

$$4\mu_y^{*} \le \sum_{x\in\mathcal{X}} \left(\alpha_r^{+}[4x+y] + \alpha_r^{-}[4x+y]\right) \le 8\mu_y^{*}$$

$$\mu_y^{*} \le \sum_{x\in\mathcal{X}} \alpha_r^{+}[4x+y], \qquad \mu_y^{*} \le \sum_{x\in\mathcal{X}} \alpha_r^{-}[4x+y]$$

$$4 \le 4\nu_{y,8}^{*} + \sum_{x\in\mathcal{X}} \nu_{y,x}^{*}$$

$$6\nu_{y,8}^{*} \le \sum_{x\in\mathcal{X}} \left(\alpha_r^{+}[4x+y] + \alpha_r^{-}[4x+y]\right)$$

$$2\nu_{y,8}^{*} = \sum_{x\in\mathcal{X}} \alpha_r^{+}[4x+y] - 2\nu_{y,9}^{*} - \nu_{y,10}^{*}$$

$$\nu_{y,10}^{*} = 2\nu_{y,x+4}^{*} + 1 - \nu_{y,x}^{*} - \alpha_r^{+}[4x+y] - \alpha_r^{-}[4x+y] \qquad \forall x \in \mathcal{X}$$

$$1 \le \sum_{b\in\mathcal{B}} \alpha_0^{+}[b] + \sum_{b\in\mathcal{B}} \tau[b] \qquad \text{(Non-triviality)}$$

Figure 4.2: Truncated differential MILP model of MANTIS_r (from [Eic17]).

characteristics may not result in a valid non-truncated differential characteristic. Since the truncated model is only abstracting each cell as 1 bit (active/inactive) instead of the real cell size of 4 bits, impossible dependencies may appear when extending the truncated characteristic to a full one. To quickly verify truncated differential characteristics output by the MILP solver, we use a non-linear differential search tool originally designed to find differential characteristics in hash functions.

This tool has been used to find differential characteristics for SHA-1 [DR06], SHA-256 [MNS11; MNS13] and SHA-512 [EMS14; DEM15], among others. Its basic search strategy is based on a guess-and-determine approach. Guided by user-chosen priorities, unrestricted bits are guessed and then the new restrictions arising from that guess are propagated to other bits. If an inconsistency is encountered, the search is reverted to a previous state, and another guess is made. Look-ahead strategies are applied to guide the search towards more promising branches in the search tree.

Due to the size of the search space, a search for differential characteristics using just this tool does not yield high-probability results in a reasonable time. However, by taking the results of the MILP model and using them to constrict which cells are active and which are not, the search space is greatly reduced. When given a truncated characteristics as input that does not contain any contradictions, the search tool finds a concrete solution in a few seconds. The search tool can however not prove that no solution exists and therefore we stop the search after a short amount of time since valid solutions are usually found quickly.

4.3 Extending one Differential Characteristic to a Family

Once one concrete characteristic has been found by the non-linear differential search tool described in section 4.2, we now want to cluster several of these characteristics together, in an attempt to improve the overall probability. One approach that was considered was to collect several characteristics from the search tool and add their probabilities accordingly. However, it was soon evident that this approach is time consuming and additionally, due to the random nature

of the search tool, there is no guarantee when and if all possible characteristics have been found and added to the family.

For the key recovery attack on MANTIS_5, all of the work finding and extending the truncated characteristic into a family of characteristics was done by hand. Of course this approach is time consuming, error prone and (due to its time constraints) not easily comparable.

Because an attack on MANTIS_6 requires to compare multiple different truncated characteristics with different properties with respect to their probability and the probability of the associated family of characteristics, a computer-aided solution was needed. The basic methodology to extend one differential characteristic to a family of differential characteristics is described in the following.

We start with one concrete solution from the non-linear differential search tool. Since such a solution was found, we now know that the truncated differential characteristic is valid and that likely, more solutions following the truncated differential characteristic exist. First, the concrete solution is inserted into our cipher structure. Afterwards, all other active cells are extended to allow every possible non-zero difference $\{\mathtt{1}, \mathtt{2}, \ldots, \mathtt{f}\}$. We keep the tweak cells constrained to one single possible difference, since testing has shown that this improves the overall probability greatly (In the case of MANTIS, this difference is almost always `a`, due to the differential properties of the MIDORI S-Box). Of course not all differences are valid for each cell, so we apply rules based on the properties of the different steps in the round function to constrict the values of cells to remove the impossible differences. In contrast to classical block ciphers, for tweakable block ciphers, the linear tweakey schedule imposes many constraints on the possible differences. In the following Δ_i denotes the set of possible differences at the input of the step function and Δ_o denotes the set of possible differences at the output. $\tilde{\Delta}_i$ and $\tilde{\Delta}_o$ denote the updated sets after the application of the propagation rules.

SubCells (S). For the S-Box step, differences that are not valid for the respective input and output difference are removed from the set. We calculate the set of possible output differences based on the current set of input differences, and perform a set intersection with the current output differences. This eliminates all output differences that are not reachable with the possible input

differences. This process is then repeated in the other direction, calculating the possible input differences based on the current set of output differences and again performing a set intersection with the current set of input differences.

To ease notation we define the function $\sigma : X \mapsto Y$, so that

$$\sigma(X) = \{\, y \in \{\mathtt{0}, \ldots, \mathtt{f}\} \mid \exists\, x \in X : \mathrm{DDT}(x, y) > 0 \,\}.$$

The function takes a set of differences X and returns a set of differences Y, which includes all differences that are possible S-Box transitions for the input differences in X. Note, that due to the involutory property of the MIDORI S-Box, $\sigma^{-1} = \sigma$. We can now describe the performed set intersections using this function:

$$\tilde{\Delta}_i = \Delta_i \cap \sigma\left(\Delta_o\right) ,$$
$$\tilde{\Delta}_o = \sigma\left(\Delta_i\right) \cap \Delta_o \,.$$

PermuteCells (P). For the state permutation step, differences are propagated linearly. We perform a set intersection between the set of differences possible at the input cell and the corresponding output cell of the PermuteCells step. This removes any differences in both cells that are not possible:

$$\tilde{\Delta}_i = \Delta_i \cap \Delta_o \,,$$
$$\tilde{\Delta}_o = \Delta_i \cap \Delta_o \,.$$

AddTweakey (A). At the AddTweakey step, difference can be introduced, canceled or changed due to differences in the tweakey. For each state cell, we take the possible input differences and XOR them with the respective difference T in the tweakey. The resulting set is then intersected with the set of possible differences at the output of the operation. This eliminates all output differences that are not reachable with the possible input differences. This process is again repeated for the backwards direction:

$$\tilde{\Delta}_i = \Delta_i \cap \left(\Delta_o + T\right) ,$$
$$\tilde{\Delta}_o = \left(\Delta_i + T\right) \cap \Delta_o \,.$$

MixColumns (M). For transitions that have a branch number of 4, all cells need to have the same value to cancel out during the multiplication with the linear matrix M. Therefore, we can just perform a set intersection of the sets of the 4 active cells in the input and output for each column of the state. This eliminates all output differences that are not reachable with the possible input differences. For transitions with branch number > 4, we need to perform the calculation of the MixColumns step for each possible combination of allowed differences, and restrict the cells based on the result of this calculation. The following description covers only the case where the branch number is equal to 4, since this is the only case needed for the families of characteristics considered in this thesis. Below, $\Delta_1, \ldots, \Delta_4$ denote the 4 active cells in the MixColumns step:

$$\tilde{\Delta}_1 = \tilde{\Delta}_2 = \tilde{\Delta}_3 = \tilde{\Delta}_4 = \Delta_1 \cap \Delta_2 \cap \Delta_3 \cap \Delta_4 \,.$$

Since these restrictions impact the set of possible differences for their neighboring operations, we perform them repeatedly, until no more changes in any state cells are observed. This results in the most unrestricted family of characteristics for a given truncated differential characteristic. However, this does not necessarily mean that this family has the highest probability. For example, the input cells in the plaintext may have a high number of possible differences, but restricting these cells to a smaller subset can improve the probability of the first round S-Box transitions greatly. Furthermore, restricting cells in further rounds can also lead to improved probabilities. For example, in MANTIS, S-Box transitions

$$\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\} \rightarrow \{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\} \rightarrow \{\mathtt{a}\}$$

over 2 rounds can be improved by restricting the first set, resulting in the 2-round transition

$$\{\mathtt{a}, \mathtt{f}\} \rightarrow \{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\} \rightarrow \{\mathtt{a}\},$$

which has a higher overall probability.

4.4 Exact Computation of Probabilities

One of the main contributions of this work is the refinement of the calculation of the probability of the family of characteristics. In the case of MANTIS, the used

MIDORI S-Box has some differential properties that allow clustering several characteristics into a more probable family of characteristics. However, the calculation of the overall probability is not that trivial, since allowing more than one possible difference for a state cell can introduce dependencies for following state cells. The main problem is that the allowed difference values for one state cell do not have to be uniformly distributed. Let's look at a quick example.

Example. Assume we have the set of allowed state values $\mathcal{A} = \{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$. If we assume a uniform distribution of the input difference set (25% each), the probability that a S-Box step maps the input difference set $\mathcal{A}$ to an output difference set of $\mathcal{A}$ again is 50%. This percentage is the result of a weighted summation of the possible transition probabilities from the DDT given in Figure 2.5a. However, due to the nature of the MIDORI S-Box, if the input distribution is more heavily weighted towards a difference of a, the overall probability improves. If we assume a probability distribution of $p(\mathcal{A}) = \{0.2, 0.4, 0.2, 0.2\}$, meaning a is twice as likely as the other three possibilities, the probability that the difference set $\mathcal{A}$ maps to itself increases to 60%.

Even if we start with a uniform distribution of differences at the plaintext input, both the SubCells and the MixColumns step can change the distribution of these probabilities. We can approximate the overall probability if we assume a uniform distribution of input differences for each step, and this was the original method used in the key recovery attack on MANTIS_5. However, a more thorough investigation shows that this estimation results in a probability that is too low, and for the attack on MANTIS_6, the probability estimation is even off by a factor that would render an attack that violates the security claims impossible. One possible method to calculate the exact probability of a family of characteristics is to enumerate every possible characteristic in the family and sum their probabilities. This approach is easy to implement, since calculating the probability of a single characteristic is easy. However, for families with a huge number of characteristics, the effort needed to enumerate the family becomes prohibitive quite fast. In this section we will now describe a method that allows us to calculate the exact probability of a family of characteristics all at once and show that the overall probability improves when compared to the basic probability estimation.

We define a state $S = \{S_0, S_1, \ldots, S_{15}\}$ to represent a full state of the cipher,

with S_i being one 4-bit state cell. For each state cell S_i, we have a difference vector Δ_i, which contains all possible differences for this state cell. We further define the distribution $p(\Delta_i) = \{p_{i,0}, p_{i,1}, \ldots, p_{i,15}\}$, where $p_{i,j}$ is the probability that a difference of j is present in the state cell i. For all variables, a version with a tilde above represents the resulting variable after one state transformation is applied.

Additionally we define the four state transformations $f \in \{\mathsf{A}, \mathsf{P}, \mathsf{S}, \mathsf{M}\}$, where $f : S \mapsto \tilde{S}$. We will now describe in detail how the probability vector is affected by each of the four state transformations.

SubCells (S). To calculate the probability for the SubCells step, we need to iterate over all the possible input differences and sum the probabilities from the DDT that allow to reach a respective output probability:

$$\tilde{p}_{i,j} = \begin{cases} \sum_{k \in \Delta_i} p_{i,k} \cdot \mathrm{DDT}\,(k,j)\,, & \text{if } j \in \tilde{\Delta}_o\,, \\ 0, & \text{otherwise.} \end{cases}.$$

The probability distribution $\tilde{p}_{i,j}$ is normalized after this step.

AddTweakey (A). For the tweak addition we will only look at a simpler case, where we only allow one specific difference T in the tweak state cell. This simpler case is all that is needed to describe the probability for the families of characteristics in this thesis, since we fix the allowed tweak difference to a:

$$\tilde{p}_{i,j} = p_{i,j+T}\,.$$

In this simple case, the AddTweakey step is just a simple permutation of the difference set and the probability vector.

PermuteCells (P). The state permutation step is also very similar to the simple case of the AddTweakey step, since the output state is just a permutation of the input state:

$$\tilde{p}_{P(i),j} = p_{i,j}\,,$$

where P is the PermuteCells permutation.

MixColumns (M). For the MixColumns step, we calculate the probability for each column $c \in 1, 2, 3, 4$. We also consider only the case where the branch number is 4, similar to the rules for propagation. A branch number of 4 for the matrix M used in MANTIS results in the property that all 4 active cells must have the same difference. Therefore, all cells that are active at the output have the same probability distribution. We consider three cases with one, two and three active cells at the input respectively. Although we only provide a description of one ordering of the active cells for each case, the rest follow the same rules with swapped cell indices. The probability distribution $\tilde{p}_{i,j}$ is again normalized after this step.

Case 1 (one active cell at the input of MixColumns):

$$\tilde{p}_{c+4,j} = \tilde{p}_{c+8,j} = \tilde{p}_{c+12,j} = p_{c+0,j} \,.$$

Case 2 (two active cells at the input of MixColumns):

$$\tilde{p}_{c+0,j} = \tilde{p}_{c+4,j} = p_{c+0,j} \cdot p_{c+4,j} \,.$$

Case 3 (three active cells at the input of MixColumns):

$$\tilde{p}_{c+12,j} = p_{c+0,j} \cdot p_{c+4,j} \cdot p_{c+8,j} \,.$$

This method also works for calculating the exact probability of the conditions used in the key recovery steps in chapter 5. All listed probabilities were calculated using this tool by performing the steps described in this section backwards from the ciphertext.

4.5 Implementation

During the course of this work we developed a tool that enables a user to experiment with a family of characteristics, restricting and extending the possible values for a state cell, while having a real-time feedback about the current probability for each step. This allows for a computer-aided search of the best family for the given characteristic.

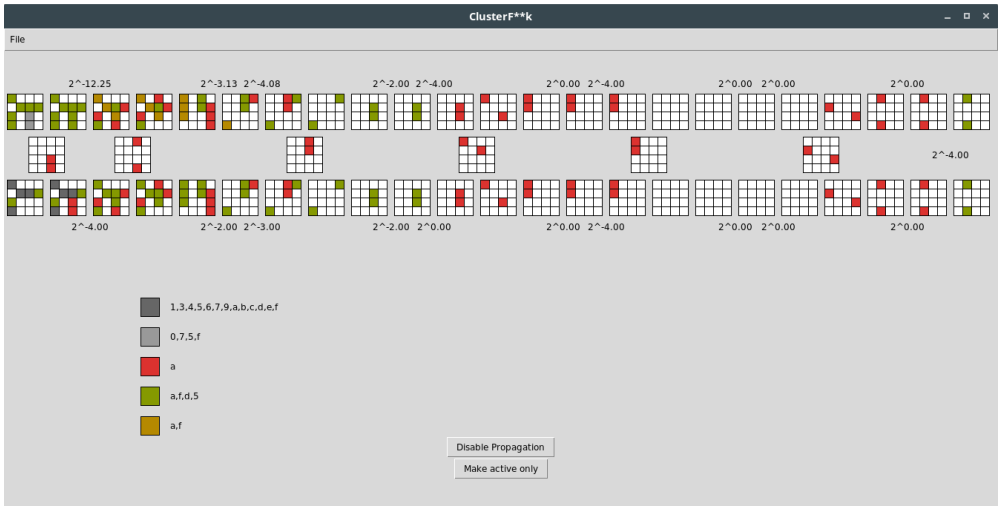


Figure 4.3: General look of the tool for probability calculation.

The tool was implemented in `python`, using a basic graphical user interface written in `TkInter`. The underlying characteristic is composed of multiple step operations (e.g., `SBOXStep`, `PermutationStep`, ...) which are arranged in the correct order to build a full block cipher like MANTIS. This design also allows for users to add other ciphers with minimal effort by just implementing new and/or reusing old step operations.

Figure 4.3 shows the general look of the tool while working on the family of characteristics for the attack on MANTIS_5. We can see the general structure of the characteristic as well as the different possibilities for state cells and the resulting probability for both the SubCells and MixColumns steps. The tool also offers an option to save and load the current family of characteristics, as well as export the family of characteristics to a LaTeX document, enabling easy generation of printable and shareable PDFs. The output for the family of characteristics illustrated in Figure 4.3 can be seen in Figure 4.4.

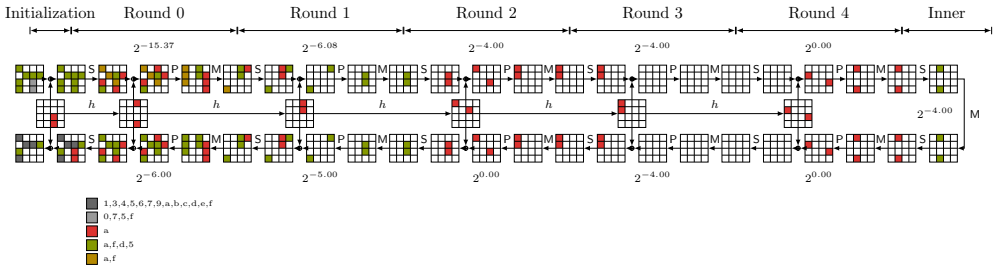


Figure 4.4: Exported PDF document for MANTIS_5.

4.6 Improved Probability Bounds for Attack on MANTIS_5

Using this new method to calculate the exact probability reveals an improvement to the previous approximation used in the original attack on MANTIS_5 [Dob+16]. For instance, the transition from the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$ to the set $\{\mathtt{a}, \mathtt{f}\}$ in round 1 has a probability that is slightly better than the result from the approximation assuming uniformly distributed input sets.

Furthermore, while rebuilding the original characteristic in the tool used to calculate the exact probability, we discovered a new, almost identical characteristic that has an even better probability than the characteristic used in the original attack. The new, improved characteristic and its probability can be seen in Figure 4.5. The calculation of the exact probability and the new structure in the first two rounds reduce the overall probability from $2^{-40.51}$ to $2^{-39.45}$. This can be used to improve the attack on MANTIS_5 described in section 3.1 further. The overall probability improved by a factor of ≈ 2, so the data complexity can also be reduced by that factor. This coincides with the observation given in section 3.2 that the number of valid pairs after the filtering process is 8 instead of the expected 4. The reduction in data complexity also reduces the time complexity for the first phases of the attack, as both the generation of the pairs and the recovery of the 44-bit subkey should be faster by a factor of 2.

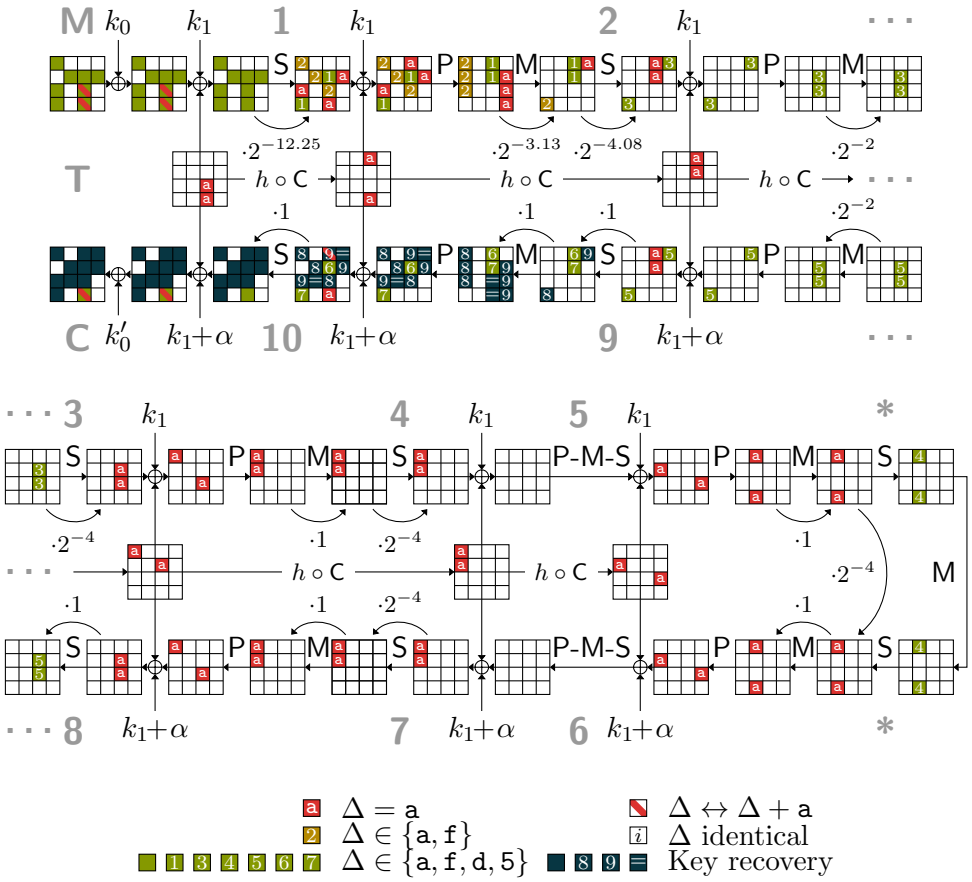


Figure 4.5: Family of differential characteristics for MANTIS_5, improved version.

5 Staged Key Recovery Attack on MANTIS$_6$

With the attack on MANTIS$_5$ being both theoretically and practically verified, the next step is analyzing the higher round versions of MANTIS. Using the same methodology used in the 5-round version however leads to several practical limitations that quickly raise the combined data and computational complexity cost above the bound of 2^{126}.

We will discuss the encountered issues and used workarounds in this chapter and present an attack on MANTIS$_6$ that recovers the secret key with a data complexity of $2^{56.72}$ and a computational complexity of $2^{56.72}$. The combined complexity of $2^{113.44}$ is well below the general bound of 2^{126} for related tweak attacks.

5.1 Extending the 5-Round Characteristics to 6 Rounds

As with the 5-round version of the attack, a characteristic with (close to) optimal probability is needed to keep the data complexity as low as possible. To generate such a characteristic we took two different approaches:

- **Extend the existing 5-round characteristic to 6 rounds:** Using the inner structure of the existing 5-round characteristic we can manually add another round to get a 6 round version of the characteristic. While this approach seems simple, most of the work is done by hand and there is no guarantee that the number of active S-Boxes is minimal. Furthermore, while the existing characteristic seems very suitable for the 5-round version there is the possibility of a completely different characteristic better suited for an attack on MANTIS$_6$.

- **Search for a new 6 round characteristic:** Because of the reasons mentioned above, we also extended the MILP model used to find a truncated characteristic for MANTIS$_6$. The MILP solver outputs a lower bound of 44 active S-Boxes for MANTIS$_6$, however all resulting characteristic have some properties that are undesirable for the other steps: There exist MixColumns transitions with branch number $\neq 4$ and there are 3 active tweak cells instead of only 2. Adding constraints for these parameters to the MILP model, we get a lower bound of 48 active S-Boxes. Upon further examination of the results it also becomes apparent that most of the resulting differential characteristic have the same inner structure as the existing 5-round characteristic.

Ultimately, both approaches resulted in the same truncated differential characteristic seen in Figure 5.1.

Using the methods and tool described in chapter 4, the truncated characteristic is developed into a family of characteristics which can be seen in Figure 5.2. We notice that the family of characteristics has more active S-Boxes in round 12 than the truncated version. This was done to improve the overall probability by allowing all possible S-Box transitions from round 11 onward for some cells, resulting in more possible differences in the round 12. We want to strike a balance between a good probability by allowing more S-Box transitions in the later rounds, a good filtering option by keeping more cells inactive in the ciphertext and a good key-recovery process by having more active cells in rounds 11 and 12. We got the best results by having exactly half of the cells in the ciphertext active and half inactive.

5.2 Generating Plaintext Pairs using Initial Structures

The family of differential characteristics in Figure 5.2 has an overall end-to-end probability of $2^{-64.19}$. Therefore we need to generate $2^{64.19}$ message pairs to have an expected number of ≈ 1 pair following the family of characteristics. The trivial approach of generating these pairs would result in $2 \cdot 2^{64.19}$ encryption oracle calls, exceeding the combined data and computational complexity bounds. Furthermore, note that the end-to-end probability of $2^{-64.19}$ is smaller than the "generic" probability of a fixed output difference of 2^{-64} and much smaller than

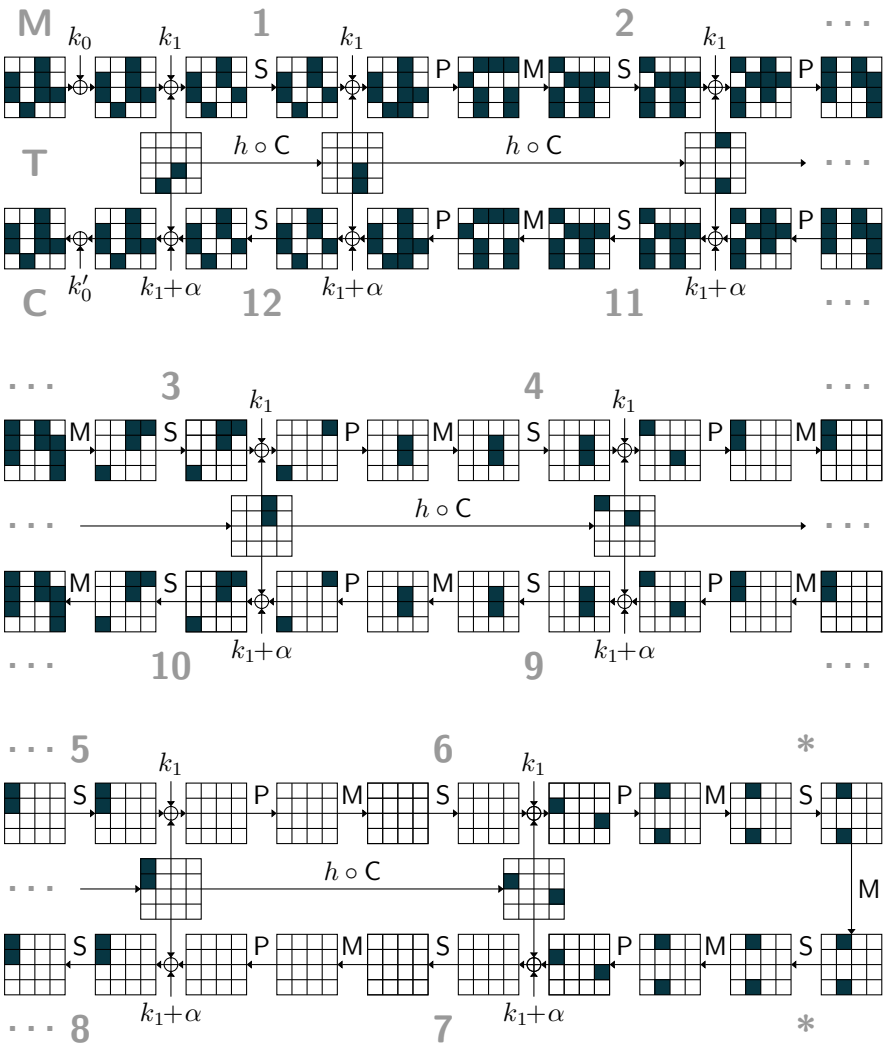


Figure 5.1: Truncated differential characteristic for MANTIS$_6$.

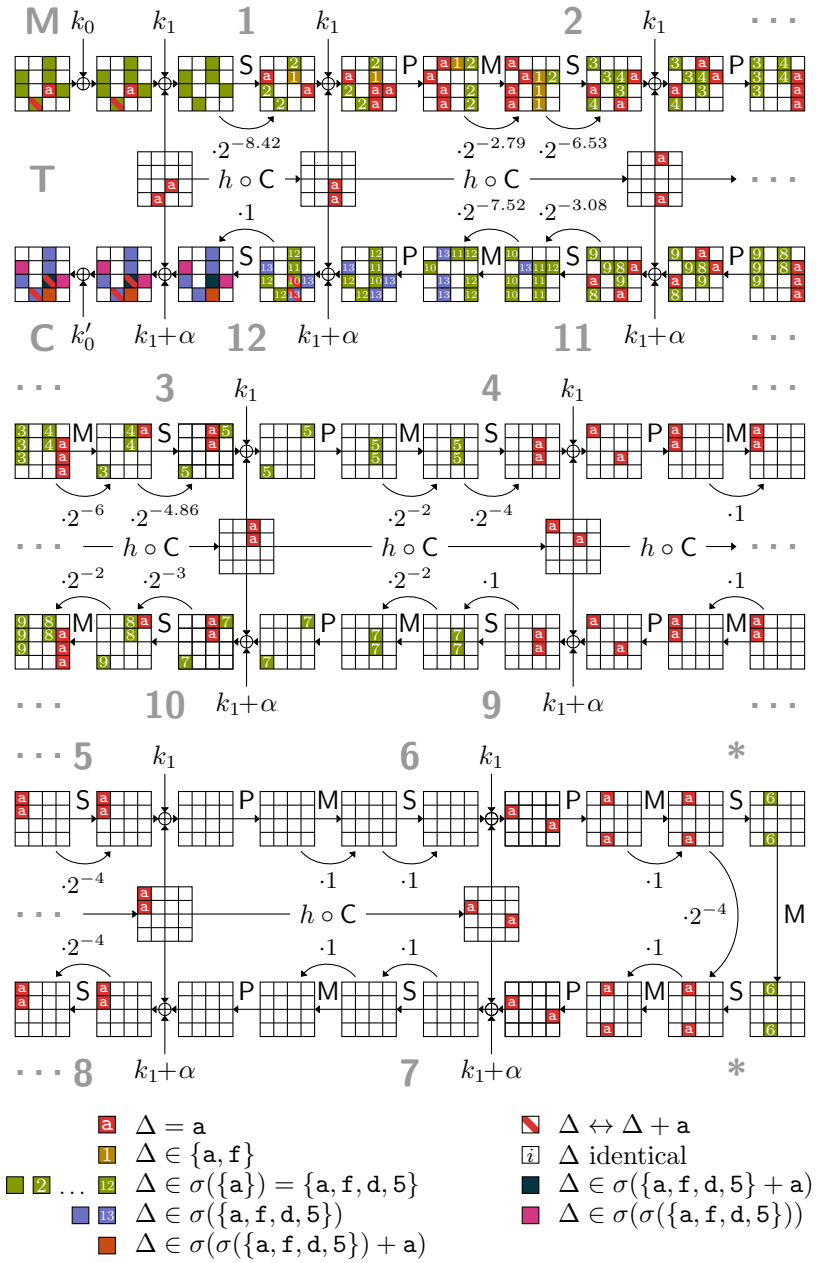


Figure 5.2: Family of differential characteristics for MANTIS$_6$.

the family of output differences allowed in our family of characteristics. This prevents a traditional key-recovery approach.

We can use the same approach as in the key recovery attack on MANTIS$_5$ to reduce the overall data complexity and the associated computational complexity by using an initial structure of plaintexts and combining them accordingly, using the set $\{5, a, d, f\}$ to our advantage.

We repeat the following process for $2^{34.19}$ base plaintext-tweak tuples. For each plaintext-tweak tuple, we query two sets of derived plaintext-tweak pairs, one for the base tweak, and one for the modified tweak with difference a in cells S_{10}, S_{13}. We vary each of the 6 active cells (■, ■) over 8 values: the base plaintext plus differences $\{0, a, f, 5, d, 8, 7, 2\}$. The second set for the modified tweak contains the same 8^6 messages. The total number of messages we query is

$$2^{34.19} \cdot 2 \cdot 8^6 = 2^{53.19}.$$

By combining all possible message pairs as outlined in Figure 5.3, the total number of message pairs generated is equal to

$$2^{34.19} \cdot 8^6 \cdot 4^6 = 2^{64.19}.$$

(a) Differences $\{a, f, d, 5\}$ (■). (b) Differences $\{0, 5, 7, f\}$ (■).

Figure 5.3: Initial structure with $8 \cdot 4$ pairs from $2 \cdot 8$ queries per cell (from [Dob+16]).

5.3 Pre-Filtering Ciphertexts for Wrong Pairs

The family of characteristics given in Figure 5.2 has a number of conditions for valid ciphertext pairs we can use to filter generated pairs before the key guessing phase:

(F1) 8 Cells $(S_0, S_1, S_3, S_5, S_7, S_9, S_{12}, S_{15})$ have a difference of zero. This condition holds with a probability of 2^{-4} per cell, resulting in an overall probability of 2^{-32}.
(F2) Cells S_2, S_6 and S_8 have to have differences in the set $\sigma(\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\})$. Additionally cell S_{13} has to have a difference in the set $\sigma(\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\})$ when accounting for the tweak difference. This condition holds with a probability of $2^{-0.299}$ per cell, resulting in an overall probability of $2^{-1.196}$.
(F3) Cell S_{10} has to have a difference in the set $\sigma(\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\} + \mathtt{a})$ when accounting for the tweak difference. The probability of this is $2^{-0.193}$.
(F4) Cells S_4 and S_{11} must have differences in the set $\sigma(\sigma(\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}))$. The probability per cell is $2^{-0.093}$, resulting in a total probability of $2^{-0.186}$.

Combining the filtering conditions (F1), (F2), (F3) and (F4), we have a filter with probability $2^{-33.58}$. We can use this filter while generating our plaintext pairs to reduce the number of overall pairs from $2^{64.19}$ to $2^{30.61}$ per repetition. This step can be implemented efficiently by grouping the ciphertexts into partitions based on the values of the relevant ciphertext cells and only combining pairs in each partition. The expected number of valid pairs per base plaintext is < 1, resulting in a complexity of $\approx 2^{53.19}$ state XOR operations per repetition.

5.4 Recovering 61 bits of key material from $k'_0 + k_1$, $k_0 + k_1$ and k_1

The first step of the attack is the recovery of 61 total bits of information about the key material. We check our key guesses against several constraints in the differential pattern of the family of characteristics in Figure 5.2.

Round 1. Guessing 24 bits of the subkey $k_0 + k_1$ allows us to compute forward until after the SubCells step in round 1. We can check our key guesses against the following conditions:

(C1) Cells S_4, S_{11} will have differences in the set $\{\mathtt{a}\}$.
(C2) Cells S_2, S_8, S_{13} will have differences in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$, and, due to the properties of the MixColumns step, these differences will be equal.
(C3) Cell S_6 will have a difference in the set $\{\mathtt{a}, \mathtt{f}\}$.

Round 12. We can additionally guess 32 bits of the subkey $k_0' + k_1$ and compute back before the last SubCells step in round 12. We can check the key guesses against the following conditions:

(C4) Cells S_2, S_8, S_{13} will have differences in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$ when computing backwards, and, due to the properties of the MixColumns step, these differences will be equal.
(C5) Cells S_4, S_{11}, S_{14} will have differences in the set $\sigma(\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\})$, and, due to the properties of the MixColumns step, these differences will be equal after accounting for the difference introduced by the tweak addition.
(C6) Cell S_6 will have a difference in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$.
(C7) Cell S_{10} will have a difference in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$, after accounting for the difference introduced by the tweak addition.

Rounds 2 and 11. Guessing keys for further rounds proves to be more computationally intensive, since keyguesses are dependent on both keyguesses for previous rounds, as well as inactive cells in the plaintext/ciphertext, for which the key also would have to be guessed. However, due to the linear nature of the MixColumns step, we can attack some more state cells:

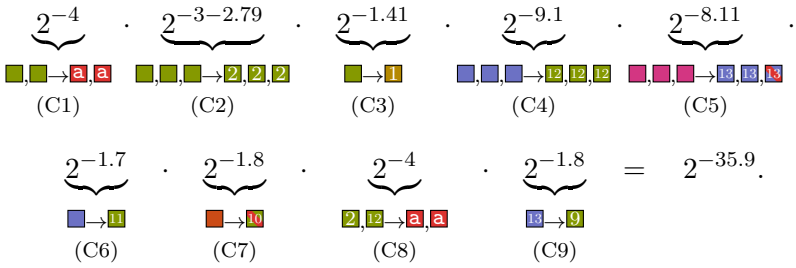
(C8) Cells S_7 after the SubCells step in rounds 2 and 11 will have differences in the set $\{\mathtt{a}\}$. Cells S_7 in round 2 and 11 depend on the keyguesses made for conditions (C2) and (C4), respectively. Furthermore, we do not need to guess the full 12 bits of k_1 that are affecting the cells S_2, S_8, S_{13} in rounds 1 and 12, since only the XOR of these state cells, and therefore the XOR of the 4-bit cell keys is relevant to the result. However, analyzing the nature of the previously guessed key bits reveals that 3 of the 4 bits in question are already predetermined as they are not linearly independent. We only need to guess one additional bit to verify this condition.
(C9) In a similar fashion, Cell S_5 in round 11 only depends on the keyguesses made for condition (C5). Again, we do not need to guess the full 12 bits of k_1 that are affecting the cells S_4, S_{11}, S_{14} in round 12, but only their 4-bit linear combination. Cell S_5 in round 11 will have a difference in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$.

The probability that a pre-filtered plaintext-tweak-ciphertext follows these conditions (C1–9) was calculated using the methods described in section 4.4

Round	Condition	Probability	Dependencies
1	(C1)	2^{-4}	-
	(C2)	$2^{-5.79}$	-
	(C3)	$2^{-1.41}$	-
12	(C4)	$2^{-9.1}$	-
	(C5)	$2^{-8.11}$	-
	(C6)	$2^{-1.7}$	-
	(C7)	$2^{-1.8}$	-
2,11	(C8)	2^{-4}	(C2),(C4)
11	(C9)	$2^{-1.7}$	(C5)

Table 5.1: Exact probabilities of conditions used for key recovery.

and is summarized in Table 5.1. The overall probability can be computed as



$$= 2^{-35.9}.$$

Due to this probability, for one single plaintext-tweak-ciphertext pair, of the 2^{61} possible sub-keys only $2^{61-35.9} = 2^{25.1}$ should follow the conditions (C1–9). Repeating this process for all $2^{30.61}$ pairs results in a total of $2^{25.1+30.61} = 2^{55.71}$ valid subkeys, reducing the key space by a factor of $2^{5.29}$. We need to repeat this process a total of 12 times to filter out the correct 61-bit subkey. These 12 repetitions increase the overall time and data complexity by a factor of $2^{3.53}$.

We compute the set of $2^{55.71}$ valid sub-keys for each repetition $r \in 1, 2, \ldots, 12$ and finally perform a set intersection of all 12 sets to calculate the correct 61-bit subkey. Using a hash-set as a data structure this can be done with a computational complexity of $2^{56.71}$ (only the first intersection is this expensive, as the set of valid keys shrinks with each intersection performed).

5.5 Recovering 43 bits of key material from $k_0' + k_1$, $k_0 + k_1$ and k_1

Using the recovered 61 bits of information about the secret key, we can further filter the plaintext pairs $i \in I_r$ that are leftover from the pre-filtering process. The probability that the right key misidentifies a pair as false positive is $2^{-35.9}$, given by the combined probability of the key-recovery conditions (C1–9). Therefore, it is likely that only pairs exactly following the family of characteristics remain after filtering the $12 \cdot 2^{30.36}$ pairs, resulting in approximately 12 pairs.

We can now use those 12 valid pairs to recover 43 bits of information about the key in two steps.

Rounds 1 and 2. We can recover 29 bits of key material by attacking the first two rounds. We want to verify that cells S_0, S_5, S_{10} of the S-Box output in round 2 follow the family of characteristics (they are in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$ and are equal). However, as mentioned in section 5.4, these cells already depend on a lot of key bits. All cells that influence the result, including the key cells, can be seen in Figure 5.4.

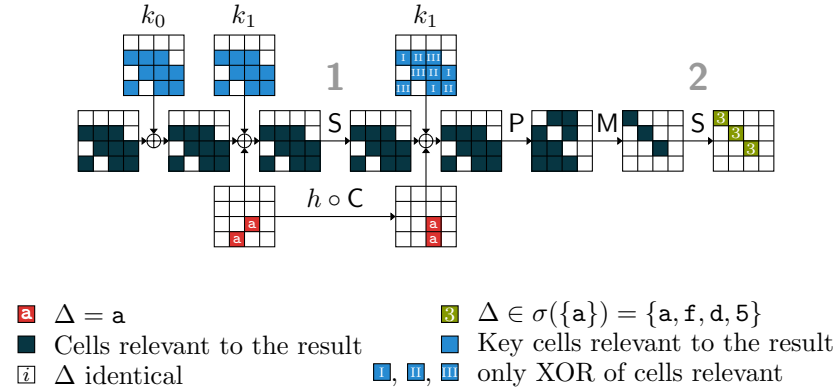


Figure 5.4: Detailed view of cells influencing the 29-bit key-recovery process.

Taking into account the previously recovered 61 bits of key material, we need to guess 29 further bits of key information to be able to compute the partial cipher

until after the S-Box layer of round 2 for each of our remaining 12 plaintext-tweak-ciphertext pairs. We can then verify that cells S_0, S_5, S_{10} follow our family of characteristics. These conditions hold with a probability of $\approx 2^{-4.25}$, or $\approx 2^{-51}$ for all 12 remaining pairs. Due to this probability, only the correct 29-bit subkey should remain.

Rounds 12 and 11. In a similar fashion, we can recover 14 more bits of information about the key by attacking the last two rounds. We want to verify that cells S_6, S_{12} of the S-Box input in round 11 follow the family of characteristics (their differences are in the set $\{\mathtt{5}, \mathtt{a}, \mathtt{d}, \mathtt{f}\}$ and are equal). Again, these cells depend on a number of key bits, and all cells and key cells that influence the result can be seen in Figure 5.5.

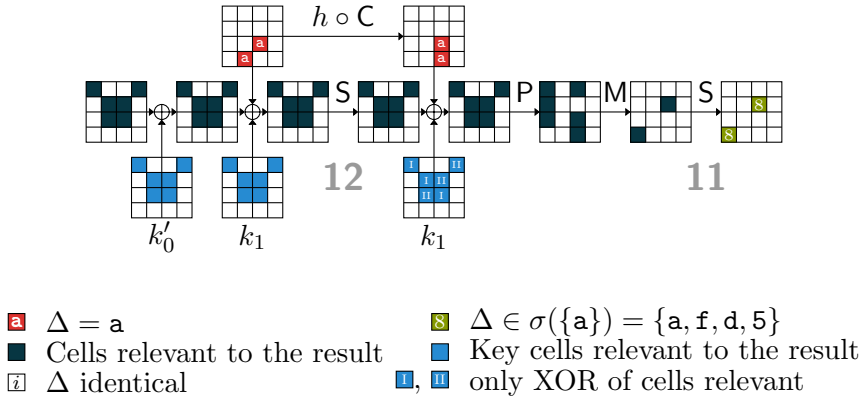


Figure 5.5: Detailed view of cells influencing the 14-bit key-recovery process.

Taking into account the previously recovered $61 + 29$ bits of key material, we need to guess 14 further bits to be able to compute the partial cipher backwards from the ciphertext until the just before the S-Box layer of round 11. We can then verify that cells S_6, S_{12} follow our family of characteristics. These conditions hold with a probability of $\approx 2^{-2.54}$, or $\approx 2^{-30.48}$ for all 12 remaining plaintext-tweak-ciphertext pairs. Due to this probability, only the correct 14-bit subkey should remain.

5.6 Recovery of k_0 and k_1, and Summary of Complexities

So far, we have recovered $61+29+14$ bits of information about the key material. This results in 104 linearly independent linear equations for k_0 and k_1.

To recover the full key, we have to guess the 24 remaining bits, resulting in a complexity of 2^{24} trial encryptions.

A summary of various complexities and probabilities can be found in Table 5.2.

Overall probability	$2^{-64.19}$
Enc. Oracle Calls per Bundle	$2^{53.19}$
Filter Probability in Ciphertexts	$2^{-33.58}$
Filtered Plaintext Pairs per Bundle	$2^{30.61}$
Recovered Key Bits in First Phase	61
Probability of Key Recovery Conditions	$2^{-35.90}$
Valid Keys per Plaintext Pair	$2^{25.10}$
Valid Keys per Repetition	$2^{55.71}$
Key Space Reduction per Repetition	$2^{5.29}$
Repetitions needed for Key Recovery	$12 \approx 2^{3.53}$
Total Data Complexity	$2^{56.72}$
Total Computational Complexity	$2^{56.72}$

Table 5.2: Summary of complexities and probabilities for the key recovery process.

5.7 Additional Comments

While searching for truncated differential characteristics, we added a number of additional constraints to the MILP model. These different constraints arose from desired properties for high-probability characteristics. In the following, we list a number of constraints that we experimented with and give a short summary of the impact these constraints have on the solutions found.

- **Restrict the MixColumns model to only include transitions with a branch number of exactly 4.** This constraint is intended to lead to

more valid truncated differentials, since transitions with a branch number of exactly 4 are easy to fulfill. However, this raises the lower bounds for active S-Boxes from 44 to 46 for MANTIS$_6$. This constraint was present in the final model used to find the truncated differential characteristics in Figure 5.1.

$$\forall y \in \mathcal{Y}, i^{\pm} \in \mathcal{R}^{\pm}:$$
$$4\mu_{i,y}^{\pm} = \sum_{x \in \mathcal{X}} \left(\mathsf{P}(\beta_i^{\pm})[4x+y] + \alpha_i^{\pm}[4x+y]\right)$$
$$\mu_{i,y}^{\pm} \leq \sum_{x \in \mathcal{X}} \mathsf{P}(\beta_i^{\pm})[4x+y], \qquad \mu_{i,y}^{\pm} \leq \sum_{x \in \mathcal{X}} \alpha_i^{\pm}[4x+y]$$

- **Restrict the number of active tweak cells to 2.** From experimentation, we observed that limiting the number of active tweak cells leads to improved probability in the final family of characteristics. This is due to the lower number of conditions that are added in each AddTweakey step. This constraint was also present in the final model, and raised the lower bound from 46 to 48 active S-Boxes in combination with the previous constraint.

$$\sum_{b \in \mathcal{B}} \tau[b] \leq 2$$

- **Only transitions with 1 or 2 active cells at the input are allowed in the last MixColumns step.** The goal of this constraint was to ensure a large number of active cells at the output of the last MixColumns step, and to improve the key-recovery process. These transitions have the condition that all cells at the output are equal, which is a good type of condition during the key-recovery process. However, the solutions found using this conditions have a much worse probability for their inner rounds, making them inferior for our attacks.

$$\forall y \in \mathcal{Y}:$$
$$1 \leq \sum_{x \in \mathcal{X}} \alpha_1^{-}[4x+y] \leq 2$$

- **The total number of active cells at the output of the last MixColumns step has to be at least 9.** Having 9 or more active cells in the last round gives us more conditions for key-recovery. But solutions following this condition have more active tweak cells, leading to much more contradictions when extending the truncated characteristic, or have a much worse overall probability.

$$9 \leq \sum_{b \in \mathcal{B}} \mathsf{P}(\beta_1^-)[b]$$

- **Only optimize rounds 2-11 (for MANTIS$_6$).** The goal of this condition was twofold. Because we have fewer rounds, we have fewer variables in the MILP model, reducing the overall time until an optimal solution is found. Additionally, when a good solution has been found, the first and last round could be modified by hand, using many of the active cells in these rounds for the key-recovery process, therefore improving the probability of the family of characteristics. However, this approach did not result in any improvement when compared to the family of characteristics in Figure 5.2.

$$\min \sum_{i^{\pm} \in \mathcal{R}^{\pm}} \quad \sum_{b \in \mathcal{B}} \alpha_i^{\pm}[b]$$

Like it was done for the practical implementation of the key-recovery attack on MANTIS$_5$, the last step (section 5.6) and most likely even the second step (section 5.5) could be implemented using a SAT-solver. In practice, modeling the cipher structure as a boolean satisfiability problem and using a high-performance SAT-solver is likely to speed up the key-recovery process when compared to enumerating the key space and performing trial encryptions.

Since the time and data complexities of this attack are not practical, we cannot verify the attack in practice. We however expect that during a practical implementation we would again encounter similar problems like differentially equivalent keys due to the second-order differential behavior of the MIDORI S-Box used in MANTIS. However, we also expect that these issues will not have a huge impact on the overall time and data complexity and that the first phase generating and pre-filtering the plaintext pairs will still dominate both the time and data requirements of the attack.

5.8 Remarks on MANTIS$_7$

MANTIS$_5$ is the most lightweight version of MANTIS, and is the only one of the MANTIS family that beats PRINCE in terms of latency and area. The designers gave explicit security claims for MANTIS$_5$ and MANTIS$_7$, and the attack presented by Dobraunig et al. [Dob+16] violates the security claims for MANTIS$_5$. In this work we present an attack on MANTIS$_6$ that violates the security claims for the full version of MANTIS. However, we expect that this kind of attack is not possible against the full MANTIS$_7$. The attack on MANTIS$_6$ already is quite close to the security claims, and one additional round increased the minimum number of active S-Boxes from 44 to 50. While this seems to be not that bad when compared to the characteristic with 48 active S-Boxes used in the attack on MANTIS$_6$ in this work, characteristics with a structure leading to the highest possible probability have more active S-Boxes. We have searched for characteristics for MANTIS$_7$ in the same way as was done for MANTIS$_6$, and the best one we found has an overall probability of $\approx 2^{-81}$. Even when using the initial structure described in section 5.2, an attack using this 7-round characteristic requires a minimum data complexity of $\approx 2^{66}$, resulting in at least the same computational complexity. This leads to a combined complexity of well over 2^{128}, rendering this kind of attack infeasible. Furthermore, the already tight bounds in the key-recovery phase of the attack on MANTIS$_6$ suggest that the increased number of pairs leftover after the pre-filtering process in the 7-round version prevent a reduction of the key space.

In conclusion, while MANTIS$_7$ seems resistant against these kind of attacks, the increased number of internal rounds means it does not offer many benefits when compared to other tweakable block ciphers such as QARMA. In the next chapter, we will discuss QARMA in detail and give a comparison to MANTIS with regards to these kind of attacks.

6 Applicability and Results for other Tweakable Block Ciphers

Over the course of this work we tried to apply the ideas of the attack to QARMA, in particular the QARMA_5 variant. Due to the success attacking MANTIS_5 and MANTIS_6, which are similar in structure, we had high hopes for similar results. However, the small differences between MANTIS and QARMA (highlighted in section 2.3) made an attack on QARMA considerably harder.

In this chapter we will talk about the applied methods, the encountered problems and attempted workarounds. We will also give some further possible enhancements to the attack method that could be explored in future work.

6.1 Extending Methods used for MANTIS to QARMA

Due to the similar structure of QARMA and MANTIS we applied the same method to find families of characteristics to QARMA_5.

We first updated the MILP model for finding differential characteristics for QARMA_5. This step includes encoding the new MixColumns layer in the mixed-integer linear program and the updated middle round. Since the MILP model for MANTIS_5 was already written using smaller building blocks for the different rounds, the insertion of another PermuteCells step in the inner round was easy.

The MixColumns layer of QARMA has a different truncated behavior (see Figure 2.6a) when compared to the MixColumns layer of MANTIS (see Figure 2.6b). We encoded these new possible transitions by starting with the XOR model proposed by Eichlseder [Eic17], which can be seen in Figure 6.1a. When comparing the truncated behavior of the MixColumns step used in QARMA to this XOR model, we can see that there are still some invalid transitions. We simply

forbid these invalid transitions explicitly by adding the following inequalities to the MILP model in Figure 4.2:

$$\forall y \in \mathcal{Y}, i^{\pm} \in \mathcal{R}^{\pm}:$$

$$\begin{aligned}
&\mathsf{P}(\beta_i^{\pm})[0+y] - \mathsf{P}(\beta_i^{\pm})[4+y] + \mathsf{P}(\beta_i^{\pm})[8+y] - \mathsf{P}(\beta_i^{\pm})[12+y] \\
&+ \alpha_i^{\pm}[0+y] + \alpha_i^{\pm}[4+y] + \alpha_i^{\pm}[8+y] - \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{5} \nrightarrow \texttt{7})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{P}(\beta_i^{\pm})[0+y] + \mathsf{P}(\beta_i^{\pm})[4+y] + \mathsf{P}(\beta_i^{\pm})[8+y] - \mathsf{P}(\beta_i^{\pm})[12+y] \\
&+ \alpha_i^{\pm}[0+y] - \alpha_i^{\pm}[4+y] + \alpha_i^{\pm}[8+y] - \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{7} \nrightarrow \texttt{5})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{P}(\beta_i^{\pm})[0+y] - \mathsf{P}(\beta_i^{\pm})[4+y] + \mathsf{P}(\beta_i^{\pm})[8+y] - \mathsf{P}(\beta_i^{\pm})[12+y] \\
&+ \alpha_i^{\pm}[0+y] - \alpha_i^{\pm}[4+y] + \alpha_i^{\pm}[8+y] + \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{5} \nrightarrow \texttt{d})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{P}(\beta_i^{\pm})[0+y] - \mathsf{P}(\beta_i^{\pm})[4+y] + \mathsf{P}(\beta_i^{\pm})[8+y] + \mathsf{P}(\beta_i^{\pm})[12+y] \\
&+ \alpha_i^{\pm}[0+y] - \alpha_i^{\pm}[4+y] + \alpha_i^{\pm}[8+y] - \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{d} \nrightarrow \texttt{5})
\end{aligned}$$

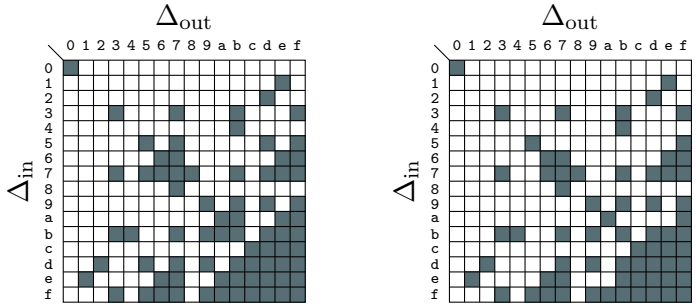
$$\begin{aligned}
&- \mathsf{P}(\beta_i^{\pm})[0+y] + \mathsf{P}(\beta_i^{\pm})[4+y] - \mathsf{P}(\beta_i^{\pm})[8+y] + \mathsf{P}(\beta_i^{\pm})[12+y] \\
&+ \alpha_i^{\pm}[0+y] + \alpha_i^{\pm}[4+y] - \alpha_i^{\pm}[8+y] + \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{a} \nrightarrow \texttt{b})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{P}(\beta_i^{\pm})[0+y] + \mathsf{P}(\beta_i^{\pm})[4+y] - \mathsf{P}(\beta_i^{\pm})[8+y] + \mathsf{P}(\beta_i^{\pm})[12+y] \\
&- \alpha_i^{\pm}[0+y] + \alpha_i^{\pm}[4+y] - \alpha_i^{\pm}[8+y] + \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{b} \nrightarrow \texttt{a})
\end{aligned}$$

$$\begin{aligned}
&- \mathsf{P}(\beta_i^{\pm})[0+y] + \mathsf{P}(\beta_i^{\pm})[4+y] - \mathsf{P}(\beta_i^{\pm})[8+y] + \mathsf{P}(\beta_i^{\pm})[12+y] \\
&- \alpha_i^{\pm}[0+y] + \alpha_i^{\pm}[4+y] + \alpha_i^{\pm}[8+y] + \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{a} \nrightarrow \texttt{e})
\end{aligned}$$

$$\begin{aligned}
&- \mathsf{P}(\beta_i^{\pm})[0+y] + \mathsf{P}(\beta_i^{\pm})[4+y] + \mathsf{P}(\beta_i^{\pm})[8+y] + \mathsf{P}(\beta_i^{\pm})[12+y] \\
&- \alpha_i^{\pm}[0+y] + \alpha_i^{\pm}[4+y] - \alpha_i^{\pm}[8+y] + \alpha_i^{\pm}[12+y] \leq 4 \qquad (\texttt{e} \nrightarrow \texttt{a})
\end{aligned}$$

Furthermore, we built a representation of the QARMA cipher for the non-linear differential search tool. The implementation of QARMA in the non-linear differential search tool was tested with the official test vectors and is used to verify the truncated differential trails output by the MILP solver.



(a) XOR model (from [Eic17]) (b) Exact model

Figure 6.1: Approximations of the truncated DDT of MixColumns used in QARMA.

6.2 Results and Workarounds

Using the methods described above, we started a search for truncated differential trails using the MILP model. The minimal number of active S-Boxes for QARMA_5 matches the number of active S-Boxes of 30 reported by Avanzi [Ava16]. However, all truncated models output by the MILP solver for 30 active S-Boxes resulted in contradictions when trying to find a concrete solution for the truncated model.

This is a result of the interaction of rotations in the MixColumns layer and the LFSR added to the tweak schedule, leading to impossible transitions over multiple rounds. This also happens for most truncated results when relaxing the minimization to 32 or 34 active S-Boxes. The small percentage of truncated differential trails leading to a full differential trail pose additional problems. Many S-Box transitions are low-probability transitions (probability of 2^{-3} instead of 2^{-2}), leading to a very poor probability for the overall trail. Furthermore, the structure of the different S-Boxes available in QARMA does not allow for easy clustering of probabilities like it was the case for MANTIS.

Observations. One possible strategy to get a high-probability trail is to use the S-Box σ_1. Its DDT includes the high-probability transition `f` $\rightarrow$ `f`. A difference of `f` means that the rotations in the MixColumns have no effect. Furthermore, if all differences are `f`, MixColumns transitions with a branch number of 4 will

always work. This allows us to build high-probability transitions for a single round of QARMA. Stitching multiple of these rounds together results in a new problem. To be able to incorporate the differences in the tweak during the AddTweakey step, the differences in the tweak also need to be `f`. However, the introduction of the LFSR in the tweak schedule does not allow for constant differences in the tweak, as the LFSR in cells $T_0, T_1, T_3, T_4, T_8, T_{11}, T_{13}$ changes the difference of `f` to a difference of `7`. This difference of `7` also has no possible transition back to `f`, further complicating connecting multiple of these high-probability rounds together. Searching for trails with a branch number of 4 results in a lower bound of 34 active S-Boxes for QARMA_5.

Another condition that could improve the possibility of finding truncated trails that lead to valid differential characteristics is to limit the number of active cells in the tweak to 2. This reduces the number of interactions between the tweak difference and the S-Box differences and also reduces the number of cells influenced by the LFSR. Searching for trails with only two active tweak cells results in a lower bound of 34 active S-Boxes for QARMA_5. Combining the two search criteria together, allowing both only MixColumns transitions with a branch number of 4 and having only two active tweak cells, results in a lower bound of 40 active S-Boxes.

A concrete differential characteristic for these combined condition can be found relatively easily, but most of the 40 S-Box transitions are still not high-probability transitions. The best probability for such a differential characteristic that we could find is 2^{-104}, which is too low for a practical key recovery attack. The differential characteristic can be seen in Figure 6.2.

6.3 Future Work

In their paper "A Security Analysis of Deoxys and its Internal Tweakable Block Ciphers", Cid et al. [Cid+17] use a similar method to find differential paths for round-reduced versions of another TWEAKEY design, Deoxys [Jea+16]. They use a MILP model of Deoxys and additionally developed a method to incorporate linear incompatibilities into the MILP model. Their model is based on the observation that the difference cancellation between the tweakey state and the round state imposes a linear relationship between the tweakey and

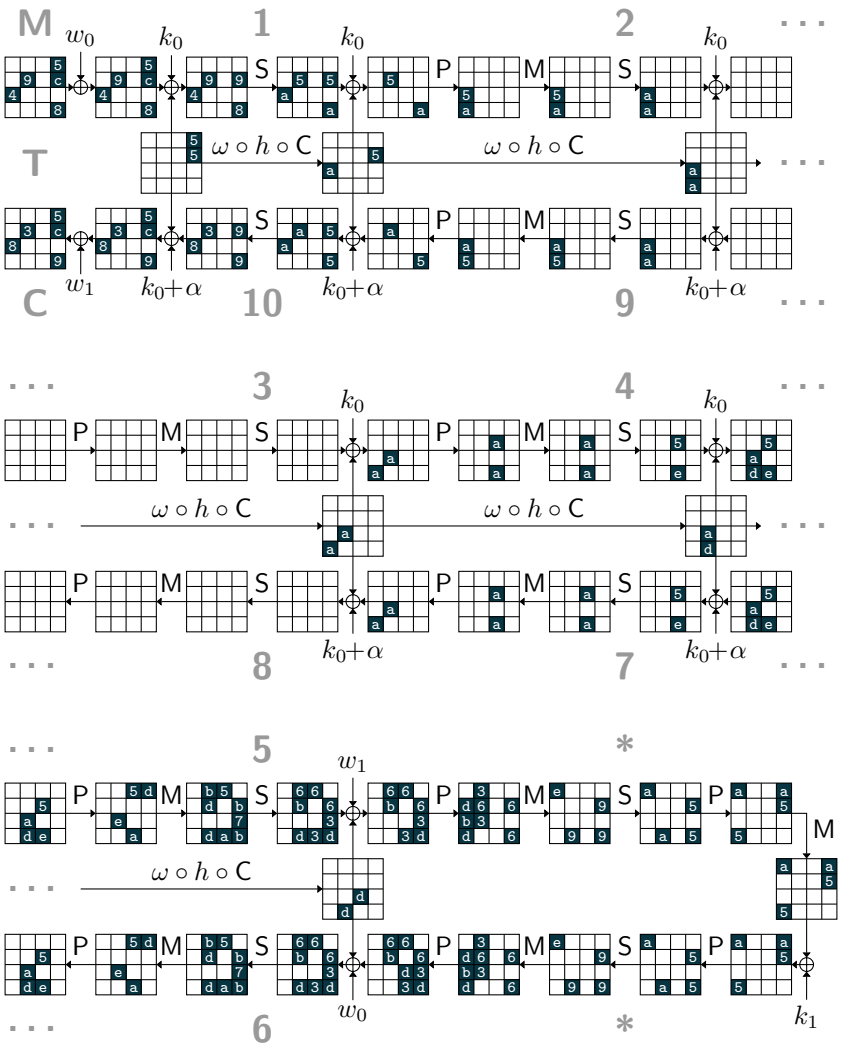


Figure 6.2: Differential characteristic for QARMA_5 (using the S-Box σ_0).

state differences, similar to the constraints discussed by Fouque et al. [FJP13] in the context of related keys. As an additional benefit, their method can also serve to provide tighter bounds for the theoretical minimum amount of active S-Boxes. Cid et al. showed that using their new model and some additional assumptions, the number of rounds to ensure 22 active S-Boxes can be reduced from 10 to 8, improving the bounds given by the designers [Jea+16].

Additionally, they also briefly discuss the approach taken in this work, where a large number of possible truncated differential trails are searched using a MILP model and then in a second stage, differential paths are searched for each truncated path until a feasible result is found. Cid et al. however found that this approach has high run-time costs and the number of differential trails grows large quite fast when compared to the improved MILP model. A similar approach could be taken for the MILP model of QARMA. Being able to incorporate linear incompatibilities into the model could lead to more and overall better differential trails.

However, the work in this thesis could also improve the attacks by Cid et al. [Cid+17]. In their work they only consider one characteristic out of those found, and clustering different characteristics together into one family of characteristics using the methods and tool described in section 4.3 could lead to improved probabilities and, in turn, attacks on more rounds of Deoxys. Due to the only recent public release of their work and the required effort, this approach was not possible to explore within this thesis, but should be considered for future research.

One further avenue to explore is the use of constraint programming (CP) based techniques instead of MILP. Sun et al. [Sun+17] show that this approach can be very efficient and even apply it to the SKINNY block cipher family to construct an 18-round attack on SKINNY-64-128. Applying these methods to MANTIS, the dedicated low-latency variant of the SKINNY family, and QARMA is something to be considered for future work.

7 Conclusion

Over the course of this thesis we have investigated tweakable block ciphers and their resistance against differential cryptanalysis. We provided an implementation for an existing key-recovery attack on MANTIS_5 and discovered some additional insights for the original theoretical attack. We then improved the methods used to find high-probability families of differential characteristics for tweakable block ciphers by creating a semi-automated toolchain.

This toolchain consists of a truncated differential model which is modeled using a mixed-integer linear program (MILP) and an efficient MILP solver, which is minimizing the number of active S-Boxes in the differential characteristic. The intermediary solutions from the MILP solver are then given to a differential search tool, which tries to find a concrete solution for the truncated solution. This filters wrong truncated solutions which have impossible constraints when extended to a full differential characteristic. Any solutions that are found are then given to a tool which extends the characteristic to a family of characteristics. This is done by allowing all possible characteristics that follow the original truncated characteristic at once, essentially allowing all differences which do not lead to contradictions for each cell. This tool then also calculates the probability for the whole family of characteristics and allows a user to manually add more constraints to single cells, further restricting or relaxing possible differences, which may lead to improved probabilities.

Using this toolchain, we improved both the data and time complexity for the original key-recovery attack on MANTIS_5 by a factor of 2. Additionally, we looked to apply the ideas of the attack to the 6-round variant of MANTIS, MANTIS_6. Using the toolchain, we found a promising family of characteristics with an end-to-end probability of $2^{-64.19}$. Although this is less than the general probability for a random MANTIS state (due to the state size of 64 bits), we presented an attack on MANTIS_6, that is able to recover the secret key using $2^{56.72}$ data and $2^{56.72}$ time complexity. The attack uses a smart choice of an

initial structure and pre-filtering to reduce the data complexity below the bounds given by the probability of the family of characteristics. The secret key is recovered using a multi-stage process spanning over multiple rounds, with the first stage recovering 61 bits of key information, the second stage recovering 43 bits of key information and the final stage recovering the missing 24 bits. Since the complexity bounds for this attack are already pretty close to the general security claims of 2^n data and 2^{126-n} time complexity, we expect that this attack will not be successful when applied to the 7-round variant, MANTIS_7.

We also tried to apply the idea of the attack and the toolchain to other modern tweakable block ciphers, in particular QARMA. QARMA is similar in structure to MANTIS and it is to be used for pointer encryption in the new ARMv8 architecture. Using the toolchain we tried to find high-probability trails for the 5-round variant, QARMA_5, but the small differences between QARMA and MANTIS proved to be more challenging to overcome than first thought. QARMA has a much better interaction between its S-Box layer and the linear mixing layer, leading to much more impossible constraints when trying to move from a truncated differential characteristic to a full differential characteristic. We were unable to find any differential characteristics with high enough probability to build any attacks for QARMA_5, however this does not mean that none exist. In fact, we give some ideas for future work that could improve the methods to find differential characteristics for QARMA.

In this thesis we have shown that although we can get lower bounds for active S-Boxes for a cipher using MILP models, these bounds can not be blindly trusted for security claims. The overall probability of differential characteristics can be greatly improved by clustering several differential characteristics together into a family of differential characteristics. Therefore it is of even greater importance to look at the interaction of the non-linear and linear layers in a cipher design with regards to differential cryptanalysis and to try to prevent clusters of differences which have high-probability transitions.

Bibliography

[Ava16] R. Avanzi. "The QARMA Block Cipher Family – Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes." In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2016), pp. 4–44. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.4-44. eprint: 2016/444 (cit. on pp. 8, 9, 13–16, 18, 55).

[Ban+15] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. "Midori: A Block Cipher for Low Energy." In: *Advances in Cryptology – ASIACRYPT 2015*. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17. eprint: 2015/1142 (cit. on pp. 8, 11, 12).

[Bei+16] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS." In: *Advances in Cryptology – CRYPTO 2016*. Ed. by M. Robshaw and J. Katz. Vol. 9815. LNCS. Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5. eprint: 2016/660 (cit. on pp. 7, 8, 11, 19).

[Bor+12] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçın. "PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications." In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by X. Wang and K. Sako. Vol. 7658. LNCS. Springer, 2012, pp. 208–225. DOI: 10.1007/978-3-642-34961-4_14. eprint: 2012/529 (cit. on p. 11).

[Bra16] D. Brash. *ARMv8-A architecture – 2016 additions*. ARM Community. 2016. URL: https://community.arm.com/processors/b/blog/posts/armv8-a-architecture-2016-additions (cit. on p. 8).

[BS90] E. Biham and A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems." In: *Advances in Cryptology – CRYPTO '90*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. LNCS. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1 (cit. on p. 2).

[Cid+17] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song. "A Security Analysis of Deoxys and its Internal Tweakable Block Ciphers." In: *IACR Transactions on Symmetric Cryptology* 2017.3 (2017), pp. 73–107. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i3.73-107 (cit. on pp. 56, 58).

[Cop94] D. Coppersmith. "The Data Encryption Standard (DES) and its strength against attacks." In: *IBM Journal of Research and Development* 38.3 (1994), pp. 243–250. DOI: 10.1147/rd.383.0243 (cit. on p. 2).

[DEM15] C. Dobraunig, M. Eichlseder, and F. Mendel. "Analysis of SHA-512/224 and SHA-512/256." In: *Advances in Cryptology – ASIACRYPT 2015*. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 612–630. DOI: 10.1007/978-3-662-48800-3_25 (cit. on p. 29).

[Dob+16] C. Dobraunig, M. Eichlseder, D. Kales, and F. Mendel. "Practical Key-Recovery Attack on MANTIS5." In: *IACR Transactions on Symmetric Cryptology* 2016.2 (2016), pp. 248–260. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i2.248-260. eprint: 2016/754 (cit. on pp. 9, 19, 21, 25, 37, 43, 52).

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4 (cit. on p. 2).

[DR06] C. De Cannière and C. Rechberger. "Finding SHA-1 Characteristics: General Results and Applications." In: *Advances in Cryptology – ASIACRYPT 2006*. Ed. by X. Lai and K. Chen. Vol. 4284. LNCS. Springer, 2006, pp. 1–20. DOI: 10.1007/11935230_1 (cit. on p. 29).

[Eic17] M. Eichlseder. "Differential Cryptanalysis of Modern Symmetric Ciphers – Draft." PhD thesis. Technical University Graz, 2017 (cit. on pp. 24, 26–28, 53, 55).

[EMS14] M. Eichlseder, F. Mendel, and M. Schläffer. "Branching Heuristics in Differential Collision Search with Applications to SHA-512." In: *Fast Software Encryption – FSE 2014*. Ed. by C. Cid and C. Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 473–488. DOI: 10.1007/978-3-662-46706-0_24 (cit. on p. 29).

[FJP13] P. Fouque, J. Jean, and T. Peyrin. "Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128." In: *Advances in Cryptology – CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, 2013, pp. 183–203. DOI: 10.1007/978-3-642-40041-4_11 (cit. on p. 58).

[GD07] V. Ganesh and D. L. Dill. "A Decision Procedure for Bit-Vectors and Arrays." In: *Computer Aided Verification – CAV 2007*. Ed. by W. Damm and H. Hermanns. Vol. 4590. LNCS. Springer, 2007, pp. 519–531. DOI: 10.1007/978-3-540-73368-3_52 (cit. on p. 25).

[Jea+16] J. Jean, I. Nikolic, T. Peyrin, and Y. Seurin. *Deoxys v1.41*. Submission to the CAESAR Competition, Round 3. 2016 (cit. on pp. 6, 56, 58).

[JNP14] J. Jean, I. Nikolić, and T. Peyrin. "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework." In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by P. Sarkar and T. Iwata. Vol. 8874. LNCS. Springer, 2014, pp. 274–288. DOI: 10.1007/978-3-662-45608-8_15. eprint: 2014/831 (cit. on pp. 6, 11).

[KR11] L. R. Knudsen and M. Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011. ISBN: 978-3-642-17341-7. DOI: 10.1007/978-3-642-17342-4 (cit. on p. 3).

[LRW02] M. Liskov, R. L. Rivest, and D. Wagner. "Tweakable Block Ciphers." In: *Advances in Cryptology – CRYPTO 2002*. Ed. by M. Yung. Vol. 2442. LNCS. Springer, 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3 (cit. on pp. 1, 5–7).

[MNS11] F. Mendel, T. Nad, and M. Schläffer. "Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions." In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, 2011, pp. 288–307. DOI: 10.1007/978-3-642-25385-0_16 (cit. on p. 29).

[MNS13] F. Mendel, T. Nad, and M. Schläffer. "Improving Local Collisions: New Attacks on Reduced SHA-256." In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, 2013, pp. 262–278. DOI: 10.1007/978-3-642-38348-9_16 (cit. on p. 29).

[Sun+17] S. Sun, D. Gerault, P. Lafourcade, Q. Yang, Y. Todo, K. Qiao, and L. Hu. "Analysis of AES, SKINNY, and Others with Constraint Programming." In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 281–306. DOI: 10.13154/tosc.v2017.i1.281-306 (cit. on p. 58).