BACHELOR'S THESIS IN COMPUTING SCIENCE

RADBOUD UNIVERSITY NIJMEGEN

Differential Cryptanalysis of the SIMON block cipher

Author:
Els de Haan
s1012212

First supervisor/assessor:
Prof. dr. ir. J.J.C. Daemen

Second assessor:
Dr. B.J.M. Mennink

Second supervisor:
MSc. J.J.P. Schoone

June 24, 2023

Abstract

In this thesis we apply differential cryptanalysis on one version of the SIMON block cipher family, namely SIMON32/64. We found a distinguishing attack that can distinguish a ten round version of SIMON32/64 from a random oracle, and conclude that SIMON32/64 reduced to (up to) ten rounds is cryptographically broken.

Contents

Chapter 1

Introduction

Cryptography is, among other things, about designing and using algorithms that can be used to prevent information from being accessible without knowing a certain key. Encryption is the conversion of information (called plaintext) to a, for third parties, inaccessible form (called ciphertext). Decryption is the the process of converting the ciphertext back to the plaintext, with the use of the key.

There are two ways of using using a key for a cipher, namely symmetric (key) cryptography and public key cryptography. In this thesis we only focus on the first category of ciphers and don't pay attention to the second category. One way of symmetric encryption is by using a block cipher (Paragraph 2.2). Feistel ciphers (Paragraph 2.3) are a special kind of block ciphers. SIMON is a Feistel cipher that was invented in 2013 by Ray Beaulieu et al. [4].

Cryptanalysis is the study of getting access to ciphertext without knowing the cryptographic key. With the help of a distinguisher (Paragraph 3.3), the security strength (Paragraph 3.4) of an cipher can be determined. Differential cryptanalysis and linear cryptanalysis are two types of cryptanalysis that are both used on block ciphers.

The block cipher family SIMON is a lightweigth blockcipher, suitable for IoT devices. It was released in 2013 ([4]) and the block cipher family consists of ten variants. In this thesis we use differential cryptanalysis on one of the variants of SIMON, namely SIMON32/64. We find a distinguisher on ten rounds of SIMON32/64 (Paragraph 5.2).

Chapter 2

Preliminaries

In this chapter, we introduce several important concepts. Paragraph 2.1 contains an explanation of the used notations in this thesis. In Paragraph 2.2 we explain the concept of a block cipher. Subsequently, in Paragraph 2.3, we address a special kind of block cipher, namely a Feistel cipher. Finally, in Paragraph 2.4, we discuss one specific family of Feistel ciphers, namely SIMON.

2.1 Notational conventions and definitions

This paragraph introduces the notational conventions and mathematical definitions that are used throughout this thesis.

2.1.1 The finite field of two elements

A field is defined as set $\mathbb{F}$ with an additive operation ($\oplus$) and a multiplicative operation ($\otimes$) that satisfy the following properties:

1. For all $a, b, c \in \mathbb{F}$, it holds that $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ and $a \otimes (b \otimes c) = (a \otimes b) \otimes c$. ($\oplus$ and $\otimes$ are associative.)

2. For all $a, b \in \mathbb{F}$, it holds that $a \oplus b = b \oplus a$ and $a \otimes b = b \otimes a$. ($\oplus$ and $\otimes$ are commutative.)

3. For all $a, b, c \in \mathbb{F}$, it holds that $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$. (Left-distributivity of $\otimes$ over $\oplus$.)

4. There exists an element $0 \in \mathbb{F}$, such that for all $a \in \mathbb{F}$, it holds that $a \oplus 0 = a$. (There exists an additive identity.)

5. There exists an element $1 \in \mathbb{F}$, such that for all $a \in \mathbb{F}$, it holds that $a \otimes 1 = a$. (There exists a multiplicative identity.)

6. For each $a \in \mathbb{F}$, there exists a $b \in \mathbb{F}$ such that $a \oplus b = 0$. (There exists an additive inverse.)
7. For each $a \in \mathbb{F}$, that is not 0, there exists a $b \in \mathbb{F}$ such that $a \otimes b = 1$. (There exists a multiplicative inverse.)

This definition is equivalent to a set S, where S with the operation $\oplus$ is a commutative group (properties 1, 2, 4, and 6 hold for $\oplus$) and $S \backslash \{0\}$ with the operation $\otimes$ is a commutative group (properties 1, 2, 5, and 7 hold for $\otimes$) and where there is distributivity (property 3) of $\otimes$ over $\oplus$.

The finite field $\mathbb{F}_2$ is the field that contains the lowest possible number of elements (two), represented as 0 and 1. These elements are known as "bits". In the field $\mathbb{F}_2$, all operations are modulo 2. The additive operation is the logical operator XOR (exclusive OR) and it is represented by the symbol $\oplus$:

$\oplus$	0	1
0	0	1
1	1	0

The multiplicative operation, also called multiplication, is the logical operator AND and it is represented by the symbol $\otimes$:

$\otimes$	0	1
0	0	0
1	0	1

2.1.2 The vector space $\mathbb{F}_2^n$

Given the field $\mathbb{F}_2$ and an integer $n > 0$, $\mathbb{F}_2^n$ is a vector space under vector addition ($\oplus$) and scalar multiplication ($\cdot$), where the scalars are elements of $\mathbb{F}_2$. This means that for all $u, v, w \in \mathbb{F}_2^n$ and for all $a, b \in \mathbb{F}_2$ it holds that:

1. $u \oplus (v \oplus w) = (u \oplus v) \oplus w$ (associativity of vector addition)
2. $u \oplus v = v \oplus u$ (commutativity of vector addition)
3. There exists a $\mathbf{0} \in \mathbb{F}_2^n$, such that $v \oplus \mathbf{0} = v$ (additive identity)
4. For every $v \in \mathbb{F}_2^n$ there exists an $-v$ such that $v \oplus (-v) = \mathbf{0}$ (additive inverse)
5. $a \cdot (b \cdot v) = (a \otimes b) \cdot v$ (compatibility of scalar multiplication and field multiplication)
6. There exists a $1 \in \mathbb{F}_2$ such that $1 \cdot v = v$ (multiplicative identity)

7. $a \cdot (u \oplus v) = a \cdot u \oplus a \cdot v$ (distributivity of scalar multiplication over vector addition)
8. $(a \oplus b) \cdot v = a \cdot v \oplus b \cdot v$ (distributivity of scalar multiplication over field addition)

A vector of three elements is denoted as (x, y, z). Take for example the vectors $v = (0, 0, 1, 1)$ and $w = (0, 1, 0, 1)$, that are both elements of $\mathbb{F}_2^4$, the vector addition

$$v \oplus w = (0, 0, 1, 1) \oplus (0, 1, 0, 1) = (0, 1, 1, 0)$$

is also an element of $\mathbb{F}_2^4$. Given a vector $u = (0, 1) \in \mathbb{F}_2^2$ and a scalar $s = 1 \in \mathbb{F}_2$, the scalar multiplication $s \cdot u = 1 \cdot (0, 1) = (0, 1)$ also is an element of $\mathbb{F}_2^2$. A vector $(x_1, x_2, x_3, \ldots, x_n)$ in $\mathbb{F}_2^n$ can also be represented as a string $x_1x_2x_3 \ldots x_n$. We use both notations. We may call vector addition XOR'ing. We use the AND operation ($\otimes$) on bit vectors, for example:

$$(0, 0, 1, 1) \otimes (0, 1, 0, 1) = (0, 0, 0, 1).$$

2.1.3 Bit strings in hexadecimal representation

A group of four consecutive bits (16 possibilities) can be represented with one hexadecimal digit: 0-F. 0000, 0001, 0010, ..., 1111 are represented respectively by 0, 1, 2, ..., F. In this thesis, if a string of digits has '0x' in front of it, it means that it is in hexadecimal representation. For example, the bit string 00101100 is 0x2C in the hexadecimal represtation.

2.1.4 Concatenation of bit strings

We use the notation $||$ for the concatenation of bit strings, for example:

$$0011||0101 = 00110101.$$

2.1.5 Bitwise rotation

The notation S^k, where $k \in \mathbb{Z}$ (the set of integers), represents the bitwise rotation operation, in which the bits of a vector in $\mathbb{F}_2^n$ are rotated k bits to the left. The total number of bits remains the same. The bits that overflow on the left side, are added on the right side. The operation $S^k \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined as $S^k(x_1, x_2, \ldots, x_n) = (x_{k+1}, x_{k+2}, \ldots, x_n, x_1, x_2, \ldots, x_k)$. For example $S^2(\underline{11}101100) = 101100\underline{11}$. If k is negative, the bits are $-k$ bits rotated to the right. For example, $S^{-2}(111011\underline{00}) = \underline{00}111011$.

2.1.6 Linear and affine maps

A map $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is linear if for all vectors $v, w \in \mathbb{F}_2^n$ and for any scalar $c \in \mathbb{F}_2$ holds:

- $f(v \oplus w) = f(v) \oplus f(w)$ (additivity)
- $f(c \cdot u) = c \cdot f(u)$ (homogeneity of degree 1)

An example of a linear map is S^k (Paragraph 2.1.5).

A map $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is affine if $f(v) = L(v) \oplus w$, where L is a linear map and $w \in \mathbb{F}_2^n$.

2.1.7 Hamming weight of bitstring

The Hamming weight of a string of bits is defined as the number of 1's that it contains.

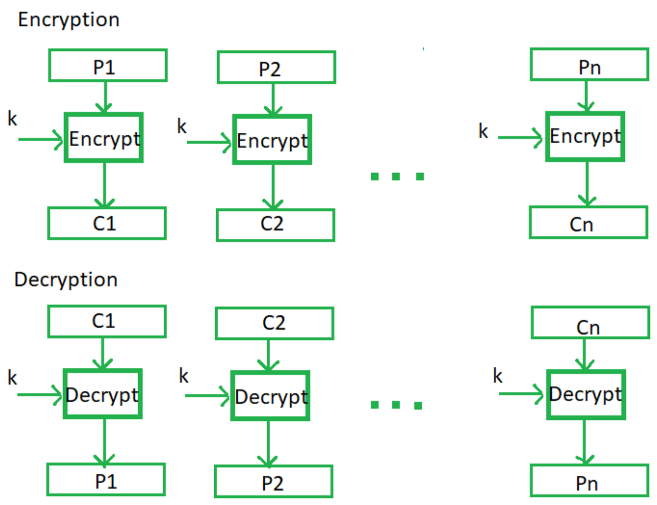
2.2 Block ciphers

A cipher is a pair of an encryption and a decryption algorithm, where decryption is the inverse of encryption. A block cipher is a symmetric-key cipher, which means that the same key is used for encryption and decryption. A block cipher takes as input a block of data of fixed length, called the *block size*, and a secret key k. The output is a block of data of the same length as the input data block.

To encrypt a plaintext that is longer than the block size, we need an algorithm that specifies how to repeatedly apply the block cipher. This is known as the mode of operation.

In case of encryption in ECB (Electronic Code Book) mode (see Figure 2.1), the plaintext is divided in blocks of fixed size and the block cipher is applied on those blocks of plaintext in parallel, always with the same secret key, resulting in blocks of ciphertext of the same lengths as the plaintext blocks. Of course, this requires the length of the plaintext to be an exact multiple of the block size. If this is not the case, bits are padded to the plaintext.

Figure 2.1: ECB mode. Figure taken from [13].



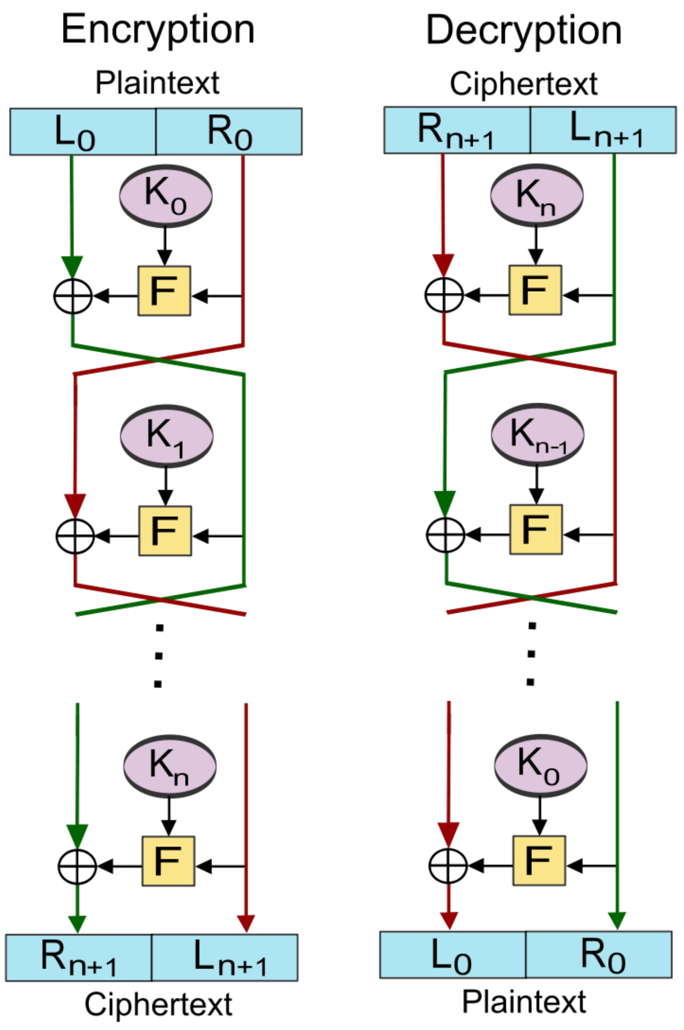
To decrypt a block of ciphertext, the block cipher gets as input the block of ciphertext and the same secret key that was used for its encryption, and the output is a block of the plaintext. In ECB mode two blocks of plaintext that are the same, will always result in the same block of ciphertext. To prevent this direct relationship, to increase the security, other modes of operation are much more commonly used nowadays.

In most block cipher algorithms, a certain invertible transformation is repeatedly applied to a state. The initial state is the input of the block cipher, the final state is the ciphertext. Each iteration is called a round and contains the same transformation. In each round, this transformation gets as input a data block and a (sub)key. These subkeys are generated using a key schedule and the given secret key. The output of the transformation is the data block that is the input in the next round. These block ciphers are called 'iterated block ciphers'. Some notable examples of block ciphers are RC5 [16], AES [15] and Blowfish [17]. Both RC5 and Blowfish are Feistel ciphers, which are a specific kind of block ciphers.

2.3 Feistel ciphers

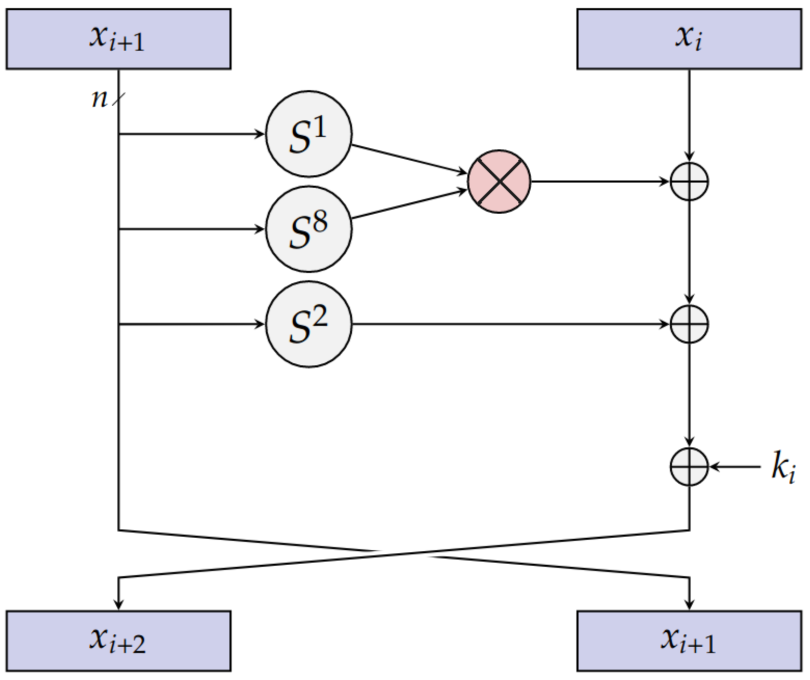
A Feistel cipher [12] is an iterated block cipher where in each round, a certain fixed function (displayed as the F blocks in Figure 2.2) takes a subkey and half of the block of data (R_0 in the first round), and then the outcome is XORed with the other half of the data block (L_0 in the first round). In the first round, the block of data is (a part of) the plaintext. In all next rounds, the block of data consists of the output of the function F of the previous round and half of the data that did not go through the F-function in the previous round (which is L_0 in the second round). In those rounds, F takes the half of the data that did not go through it previously, and a subkey. The outcome is XORed with the outcome of the round function of the previous round. After a certain amount of rounds (depending on the algorithm), it results in the ciphertext. An advantage of a Feistel cipher is that decryption is as easy as encryption. It works the same, only the order in which the subkeys are used is reversed.

Figure 2.2: Structure of a Feistel cipher. Figure taken from [3].



2.4 SIMON

Figure 2.3: The round function of SIMON. Figure taken from [4].



One example of Feistel ciphers is the family SIMON. It was designed by Ray Beaulieu et al. [4] and has been released in 2013. It is a lightweight block cipher, designed for devices with relatively little computational power (IoT devices [5]). There are ten variants of SIMON, each with a different combination of block size/key size, namely: 32/64, 48/72, 48/96, 64/96, 64/128, 96/96, 96/144, 128/128, 128/192, 128/256. A secret key consists of multiple “key words”. The ten variants differ in the number of rounds and in the number of key words, as shown in Table 2.1. These key words are the subkeys for the first rounds and are used to produce subkeys for the next rounds.

Like in other Feistel ciphers, in each round, the round function of SIMON takes half of a data block and a subkey and the outcome of the round function is XORed with the other half of the data block. As can be seen in Figure 2.3, the round function of SIMON contains a bitwise AND function that combines this data rotated one bit to the left, and the same data rotated eight bits to the left. The result is subsequently XORed with the data rotated two bits to the left and with the subkey. Just as in other Feistel ciphers, the result of the round function is XORed with the other half of the data block, and the result of this is the data input for the next round function.

Table 2.1: Parameters of SIMON.

block size	**key size**	**word size**	**number of key words**	**number of rounds**	**sequence**
2n	**mn**	**n**	**m**		
32	64	16	4	32	z_0
48	72	24	3	36	z_0
	96		4	36	z_1
64	96	32	3	42	z_2
	128		4	44	z_3
96	96	48	2	52	z_2
	144		3	54	z_3
128	128	64	2	68	z_2
	192		3	69	z_3
	256		4	72	z_4

Next, we give an example of one round SIMON32/64 with the subkey=0x0100 and the data block=0x65656877. The round function takes the subkey and the first (left) half of the data block:

		0110 0101 0110 0101	(0x6565 in binary)
	$S^1(a_{10})$:	1100 1010 1100 1010	(0x6565 rotated)
$\otimes$	$S^8(a_{10})$:	0110 0101 0110 0101	(0x6565 rotated)
		0100 0000 0100 0000	(result of AND)
$\oplus$	$S^2(a_{10})$:	1001 0101 1001 0101	(0x6565 rotated)
$\oplus$	key :	0000 0001 0000 0000	(first subkey 0x0100)
		1101 0100 1101 0101	(result of XOR: 0xD4D5)

The output of this round is the second half of the plaintext, 0x6877, concatenated with the result of the round function, 0xD4D5, which results in 0x6877D4D5.

Each round takes a different subkey. As can be seen in Table 2.1, the secret key consists of 2, 3 or 4 key words. These key words are the subkeys for the first 2, 3 or 4 rounds respectively. For the other rounds the subkeys are generated by using the subkeys of earlier rounds. For SIMON with word size n and m key words, the key schedule for producing those subkeys is as follows:

$$k_{i+m} = \begin{cases} c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}(k_{i+1})) & \text{if } m = 2; \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}(k_{i+2})) & \text{if } m = 3; \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}(k_{i+3}) \oplus k_{i+1}) & \text{if } m = 4. \end{cases}$$

Here c stands for a constant $2^n - 4 = $ 0xFF...FC and $(z_j)_i$ is the i-th bit of z_j, a certain sequence that is predetermined by the authors of SIMON (see Table 2.1). The $I(= S^0)$ stands for the identity function $I(x) = x$. As example, let 0x1918 1110 0908 0100 be a secret key for SIMON32/64. The first subkeys are then $k_0 =$ 0x0100, $k_1 =$ 0x0908, k_2=0x1110 and k_3=0x1918. The next subkey k_4 is then calculated like this:

	k_3:	0001 1001 0001 1000
S^{-3}		0000 0011 0010 0011
$\oplus$	k_1:	0000 1001 0000 1000
	I :	0000 1010 0010 1011
$\oplus$	$S^{-1}I$:	1000 0101 0001 0101
		1000 1111 0011 1110
$\oplus$	k_0:	0000 0001 0000 0000
$\oplus$	$(z_0)_0$:	1
$\oplus$	c:	1111 1111 1111 1100
		0111 0001 1100 0011

Thus, subkey k_4 is 0x71C3 here. Note that this key schedule only contains the linear operations XOR, bitwise rotation and constant addition, which means that the key schedule is affine.

Chapter 3

Security of a block cipher

Cryptanalysis is, among other things, studying the security of ciphers by looking for ways to find some connection between a ciphertext and its plaintext when the key is unknown. An important concept in cryptanalysis is a distinguisher. This chapter explains what a distinguisher is and how it can be used. Furthermore, this chapter shows how, with the help of a distinguisher, the security strength of a block cipher can be determined.

3.1 Security claim

The adversary can send a certain amount of queries to the cipher under attack, which is called the "online complexity" denoted with M, and he can do a certain amount of calculations, which is called the "offline complexity" denoted with N [9]. A security claim for a cryptographic cipher claims that there does not exist an attack with a higher success probability for a certain M and N than a certain predetermined probability. If an attack is found that has an higher succes probability than the probability in the security claim for the cipher, the cipher is considered "broken". To "repair" the cipher, either the security claim has to be weakened or the cipher has to be strengthened.

3.2 Exhaustive key search

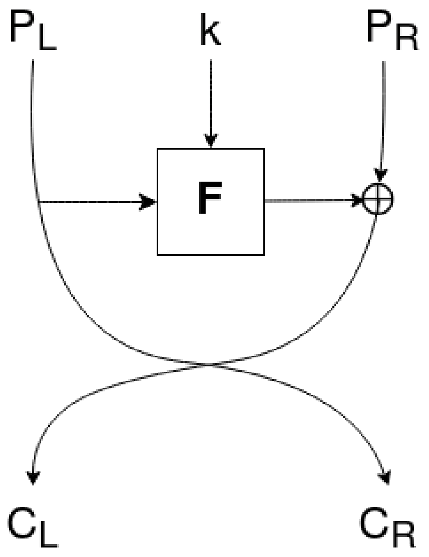
The most basic attack against an encryption algorithm with a secret key is *exhaustive key search*. The attacker has access to a ciphertext and the corresponding plaintext and tries to find the right key by systematically trying to decrypt the ciphertext with every possible key until it results in the right plaintext. For a key of n bits, there are 2^n possible keys. On average the amount of attempts that it takes to find the right key, is half of the amount of all possible keys, so $\frac{2^n}{2} = 2^{n-1}$ attempts. In case of SIMON32/64, the key size is 64 bits. So, exhaustive key search on SIMON32/64 will cost $\frac{2^{64}}{2} = 2^{63}$ attempts on average. The online complexity $M = 1$ and the offline complexity $N = 2^{63}$ in this case.

The default security claim for a cipher is that there is no attack more efficient than exhaustive key search to find the cryptographic key or to distinguish the cipher from a random permutation. So, if a block cipher is published without a specified security claim, it is considered broken if an attack is found that is significantly better than exhaustive key search.

3.3 Distinguishers

For a block cipher to be good, it should be indistinguishable from a random permutation if the key is unknown but fixed. Such a block cipher is called a pseudorandom permutation (PRP). A distinguishing attack is an attempt to distinguish a block cipher from a random permutation. A distinguisher tests on the basis of one or multiple queries which of the possibilities (a certain block cipher vs. a random permutation) is the most likely.

Figure 3.1: One round of a Feistel cipher.



To see that one round of Feistel (see Figure 3.1) with secret key k is easily distinguishable from a random permutation (RP), and thus insecure, take a distinguisher $\mathcal{A}$, consisting of the following algorithm:

1. Send a query $\mathrm{P} = \mathrm{P_L} \| \mathrm{P_R}$ (plaintext), which gives the result $\mathrm{C} = \mathrm{C_L} \| \mathrm{C_R}$ (ciphertext).
2. Check whether $\mathrm{P_L} = \mathrm{C_R}$.
3. Output $\quad \mathcal{A} = 1 \quad$ if $\mathrm{P_L} = \mathrm{C_R}$;
   or $\quad \mathcal{A} = 0 \quad$ if $\mathrm{P_L} \neq \mathrm{C_R}$.

For a random permutation on strings of $\frac{n}{2}$ bits, there are $2^{\frac{n}{2}}$ possibilies, and each possibility occurs with the same probability. Given that the plaintext $(\mathrm{P_L}||\mathrm{P_R})$ and the ciphertext $(\mathrm{C_L}||\mathrm{C_R})$ are both n bits long, the probabilities that $\mathrm{P_L} = \mathrm{C_R}$ for a one round Feistel and a random permutation respectively are:

- $\Pr(\mathrm{P_L} = \mathrm{C_R} \mid \text{Feistel}) = 1$, because for one round of a Feistel cipher, the 'left' half of the input is always the same as the 'right' half of the output.
- $\Pr(\mathrm{P_L} = \mathrm{C_R} \mid \text{RP}) = \frac{1}{2^{\frac{n}{2}}}$, because $\mathrm{P_L}$ and $\mathrm{C_R}$ both consist of $\frac{n}{2}$ bits.

From these probabilities it is clear that if $\mathrm{P_L} = \mathrm{C_R}$ is true, then, with a very high probability, the examined function is a Feistel cipher and not a random permutation.

We can calculate the advantage ([9]) of this distinguisher. The advantage of a certain pseudorandom permutation X is calculated as follows:

$$\mathrm{Adv}_{\mathcal{A}} = \Big|\Pr[\mathcal{A} = 1 \mid \mathrm{X}] - \Pr[\mathcal{A} = 1 \mid \mathrm{RP}]\Big|.$$

In this case, the advantage is thus:

$$\mathrm{Adv}_{\mathcal{A}} = \Big|\Pr[\mathcal{A} = 1 \mid \text{Feistel}] - \Pr[\mathcal{A} = 1 \mid \mathrm{RP}]\Big| = \Big|1 - \frac{1}{2^{\frac{n}{2}}}\Big|.$$

So for a large n, the advantage that the function is a one round Feistel in case of $\mathrm{P_L} = \mathrm{C_R}$ is indeed very high. For SIMON32/64, $n = 32$, so $\mathrm{Adv}_{\mathcal{A}} = \left|1 - \frac{1}{2^{16}}\right| \approx 1$. So, a one round Feistel cipher is broken, because there is a distinguishing attack that is significantly better than exhaustive key search. The advantage is used to calculate the security strength.

3.4 Security strength

Given the online complexity M, the offline complexity N and the success rate p, in case of a key-recovery-attack or the advantage $\mathrm{Adv}_{\mathcal{A}}$ in case of a distinguisher $\mathcal{A}$, an encryption algorithm has a security strength [10] of s, if there are no attacks for which holds that

$$s \leq \min\Big(\log_2\Big(\frac{N+M}{p}\Big), \log_2\Big(\frac{N+M}{\mathrm{Adv}_{\mathcal{A}}}\Big)\Big).$$

An attack gives thus an upper bound for the security strength of a cipher. To consider the security strength in case of a key-recovery-attack, take for example exhaustive key search on SIMON32/64, which has a key size of 64 bits. Given the online complexity N and the offline complexity $M = 1$, the success rate p is $\frac{N}{2^{64}}$. This gives us

$$\log_2\left(\frac{N+M}{p}\right) = \log_2\left(\frac{N+1}{\frac{N}{2^{64}}}\right) = \log_2\left(2^{64}\cdot\left(1+\frac{1}{N}\right)\right) = 64+\log_2\left(1+\frac{1}{N}\right).$$

For a large N, $\frac{1}{N}$ is negligible, so we get 64. This means that the security strength of SIMON32/64 is at most 64 bits.

To determine the security strength in case of a distinguishing attack, let us consider the distinguisher $\mathcal{A}$ in the previous paragraph (3.3). Using that the advantage of the distinguisher $\mathcal{A}$ is 1, and given that $M = 1$ and $N = 1$, we find

$$\log_2\left(\frac{N+M}{\mathrm{Adv}_{\mathcal{A}}}\right) = \log_2\left(\frac{1+1}{1}\right) = 1.$$

The security strength of a one round Feistel cipher is thus at most 1 bit.

Chapter 4

Differential cryptanalysis

Cryptanalysis is the study of the relationship between a message in plaintext and its corresponding ciphertext, in order to deduce information about the encrypted messages without knowing the cryptographic key. Differential cryptanalysis ([11]) is one of the forms of cryptanalysis that is most commonly applied to block ciphers. It studies how the differences between plaintexts influence the differences between the corresponding ciphertexts.

A combination (a', b') of $a' \in \mathbb{F}_2^n$, a difference between two inputs, and $b' \in \mathbb{F}_2^n$, a difference between two outputs, is called a differential. Given a certain input difference a' and a certain output difference b', the probability that for a random input a the encryption of the input a added to the encryption of the result of the input a added to the input difference a', results in the output difference b' is called the "differential probability". The mathematical definition is:

$$\mathrm{DP}_f(a', b') = \#\{a \in \mathbb{F}_2^n \mid f(a) \oplus f(a \oplus a') = b'\}/2^n.$$

The differential probabilities of a linear (XOR) function, an AND function, and a constant addition function are discussed below. In the end, it is explained how to calculate the differential probability of a differential trail. Given a DP of 2^{-w}, w is called the weight of the differential.

4.1 Linear functions

A function is linear if $f(a \oplus a') = f(a) \oplus f(a')$, for all a and a', so the difference in the input will directly determine the difference in the output. The differential probability of a linear function is then either 1 or 0, as shown below.

$$\begin{aligned}\mathrm{DP}_f(a', b') &= \#\{a \in \mathbb{F}_2^n \mid f(a) \oplus f(a \oplus a') = b'\}/2^n \\ &= \#\{a \in \mathbb{F}_2^n \mid f(a) \oplus f(a) \oplus f(a') = b'\}/2^n \\ &= \#\{a \in \mathbb{F}_2^n \mid f(a') = b'\}/2^n \\ &= \begin{cases} 1 \text{ if } f(a') = b'; \\ 0 \text{ otherwise.} \end{cases}\end{aligned}$$

So, when a function f is linear and $f(a') = b'$, then the differential probability of (a', b') is 1.

4.2 The AND-function

If a function contains a multiplication, also referred to as AND, it is more complex to calculate the differential probability. That is because of the fact that the input difference does not predict with certainty what the output difference will be. Take for example the function $f\colon \mathbb{F}_2 \times \mathbb{F}_2 \mapsto \mathbb{F}_2$, $f(a, c) = a \otimes c$ (multiplication over $\mathbb{F}_2$). Lets us consider the bits a, a^*, c and c^*, and the input differences $a' = a \oplus a^*$ and $c' = c \oplus c^*$. Given the outputs $b = f(a, c) = a \otimes c$ and $b^* = f(a^*, c^*) = a^* \otimes c^*$, the difference of these outputs is

$$b' = b \oplus b^* = a \otimes c \oplus a^* \otimes c^*.$$

If a' and c' are both 0, then b' will always be 0, as can be seen below. Note that $a' = 0$ implies $a = a^*$, and c' implies $c = c^*$. This gives two possibilities for a and a^*, namely $a = a^* = 0$ or $a = a^* = 1$ and two possibilities for c and c^*, namely $c = c^* = 0$ or $c = c^* = 1$, so four possibilities in total.

a	$\otimes$	c	$\oplus$	a^*	$\otimes$	c^*	$=$	b'
0	$\otimes$	0	$\oplus$	0	$\otimes$	0	$=$	0
0	$\otimes$	1	$\oplus$	0	$\otimes$	1	$=$	0
1	$\otimes$	0	$\oplus$	1	$\otimes$	0	$=$	0
1	$\otimes$	1	$\oplus$	1	$\otimes$	1	$=$	0

So, $\mathrm{DP}_f((0, 0), 0) = 1$ and $\mathrm{DP}_f((0, 0), 1) = 0$.

However, when a' or c' is 1, then b' could be 0 or 1. All possibilities are shown below.

If $a' = 0$ and $c' = 1$ (results are similar if $a' = 1$ and $c' = 0$), we get:

a	$\otimes$	c	$\oplus$	a^*	$\otimes$	c^*	$=$	b'
0	$\otimes$	0	$\oplus$	0	$\otimes$	1	$=$	0
0	$\otimes$	1	$\oplus$	0	$\otimes$	0	$=$	0
1	$\otimes$	0	$\oplus$	1	$\otimes$	1	$=$	1
1	$\otimes$	1	$\oplus$	1	$\otimes$	0	$=$	1

So, $\mathrm{DP}_f((0,1),0) = \frac{1}{2}$ and $\mathrm{DP}_f((0,1),1) = \frac{1}{2}$.

If $a' = 1$ and $c' = 1$, we get:

a	$\otimes$	c	$\oplus$	a^*	$\otimes$	c^*	$=$	b'
0	$\otimes$	0	$\oplus$	1	$\otimes$	1	$=$	1
0	$\otimes$	1	$\oplus$	1	$\otimes$	0	$=$	0
1	$\otimes$	0	$\oplus$	0	$\otimes$	1	$=$	0
1	$\otimes$	1	$\oplus$	0	$\otimes$	0	$=$	1

So, $\mathrm{DP}_f((1,1),0) = \frac{1}{2}$ and $\mathrm{DP}_f((1,1),1) = \frac{1}{2}$.

A difference distribution table is a table which contains the differential probabilities of the combination of all possible input differences and output differences. From these results, the difference distribution table for this function can be obtained:

b' / (a', c')	0	1
(0, 0)	1	0
(0, 1)	$\frac{1}{2}$	$\frac{1}{2}$
(1, 0)	$\frac{1}{2}$	$\frac{1}{2}$
(1, 1)	$\frac{1}{2}$	$\frac{1}{2}$

So, if at least one of the input bit differences is one, the differential probability is always $\frac{1}{2}$. In these cases, the AND is called an 'active AND'.

4.3 Constant addition functions

For a constant addition function, such as the addition of a key, $f(x) = x \oplus k$, the constant is not relevant for a differential probability. That is because a constant addition function is applied to both of the plaintexts. By XORing those results, the constant falls away, because it occurs twice, as shown below:
From $f(a) = a \oplus k$ and $f(a \oplus a') = a \oplus a' \oplus k$ it follows that
$f(a) \oplus f(a \oplus a') = (a \oplus k) \oplus (a \oplus a' \oplus k) = a'$.

$$\begin{aligned}
\text{So}\quad \mathrm{DP}_f(a', b') &= \#\{a \in \mathbb{F}_2^n \mid f(a) \oplus f(a \oplus a') = b'\}/2^n \\
&= \#\{a \in \mathbb{F}_2^n \mid a' = b'\}/2^n \\
&= \begin{cases} 1 & \text{if } \ a' = b'; \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

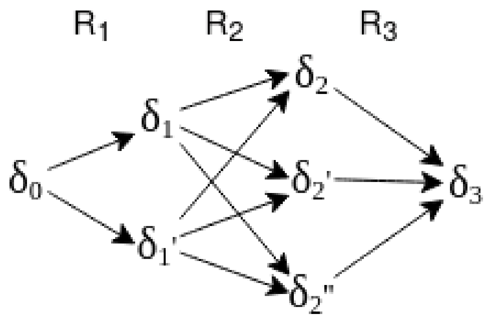
4.4 Differential trail

A differential (δ_0, δ_1) over one round is called a round differential. A sequence of round differentials $(\delta_0, \delta_1), (\delta_1, \delta_2), \dots (\delta_{n-1}, \delta_n)$ over consecutive rounds is a differential trail. The differential probability of the differential trail $(\delta_0, \delta_1, \delta_2, \dots, \delta_{n-1}, \delta_n)$ is the probability that starting with δ_0 as input in round 1, the output of round 1 is δ_1, the output of round 2 is δ_2, and so on. It can be estimated by the product of the differential probability of each round differential:

$$\mathrm{DP}(\delta_0, \delta_1, \delta_2, \dots, \delta_{n-1}, \delta_n) \approx \prod_{i=0}^{n-1} \mathrm{DP}(\delta_i, \delta_{i+1}).$$ [11]

For a differential (δ_0, δ_n) over n rounds there often exist multiple differential trails. See for example Figure 4.1 were there are six different trails for the differential (δ_0, δ_3) over three rounds, marked by R_1, R_2 and R_3.

Figure 4.1: Multiple differential trails for a differential over three rounds.



If we define Q as the set of all trails for a differential (δ_0, δ_n), the differential probability of (δ_0, δ_n) is the sum of the DPs of all trails in the set Q:

$$\mathrm{DP}(\delta_0, \delta_n) = \sum_{q \in Q} \mathrm{DP}(\mathrm{q}).$$ [11]

If we want to calculate the DP of a differential over many rounds, there are often too many trails to calculate the DP for them all individually. However, given a trail with relatively high DP, the DP of the surrounding differential is often not significantly higher. The DP of a certain trail is thus a lower bound for the DP of the surrounding differential.

Chapter 5

Differential distinguisher on SIMON

In this chapter we show a differential trail on ten rounds of SIMON. In Paragraph 5.1 we assume that there is no dependency between the bits of a resulting bit string after multiplication. In reality however, there is a dependency between them, as we show in Paragraph 5.2.

5.1 Assuming independent ANDs

In this paragraph we will show a differential trail on ten rounds of SIMON32. We assume that in case of bitwise multiplication, the bits in the result are determined independently of each other.

For brevity, when we have input differences u' and v', in this chapter we write $u' \otimes v'$ for $u \otimes v \oplus u^* \otimes v^*$, where $u^* = u \oplus u'$ and $v^* = v \oplus v'$. u and v can take any possible value in $\mathbb{F}_2^n$, so the expression can have multiple outputs. We choose one of the outputs and give the probability of this output. When we have input differences u' and v', in this chapter we write $u' \oplus v'$ for $u \oplus v \oplus u^* \oplus v^*$, where $u^* = u \oplus u'$ and $v^* = v \oplus v'$. u and v can take any possible value in $\mathbb{F}_2^n$. Because this is a linear operation (see Paragraph 4.1), it does not have multiple outputs.

Let $a_0||b_0$ be the input difference. If $a_0 =$ 0000 0000 0000 0001 and $b_0 =$ 0000 0000 0000 0100, then the difference after the first round will be $a_1||b_1$, where $b_1 = a_0$. We determined a_1 as follows:

$$\begin{array}{rrll}
 & S^1(a_0): & 0000\ 0000\ 0000\ 0010 & \\
\otimes & S^8(a_0): & 0000\ 0001\ 0000\ 0000 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_0 = \frac{1}{4}) \\
 & S^2(a_0): & 0000\ 0000\ 0000\ 0100 & \\
\oplus & b_0: & 0000\ 0000\ 0000\ 0100 & \\
\hline
 & a_1: & 0000\ 0000\ 0000\ 0000 &
\end{array}$$

The operation $S^1(a_0) \otimes S^8(a_0)$ gives four possible outcomes, namely:

0000 0000 0000 0000;
0000 0000 0000 0010;
0000 0001 0000 0000;
0000 0001 0000 0010.

For the differential trail, we may focus on one of these outcomes. For each bit that is 1, two ANDs could be made active (less if they depend on each other), whereas 0 bits do not make ANDs active, so they do not increase the number of possible outcome differences. For the whole trail, it is not garanteed that this is always the best choice, because in some cases more ANDs are dependent on each other than in other cases, which could give less possibilities in the end. In order to keep the number of possibilities in each round as small as possible, we take the outcome with the least number of bits that are 1, which is 0000 0000 0000 0000 in this case. The probability that it will give this outcome is $\frac{1}{4}$. This is because there are two active ANDs, and for each of these ANDs, the probability that the resulting bit will be 0, is $\frac{1}{2}$. Here we assume that these probabilities are independent of each other. This is explained in Paragraph 4.2 about the AND-function. For two ANDs, the probability that both will result in 0 is thus $(\frac{1}{2})^2 = \frac{1}{4}$.

For the next rounds, it works in a similar way. For each round, the option with the smallest number of 1-bits is chosen and the probability for this option is given. In general, $b_i = a_{i-1}$, except for $i = 0$.

$$\begin{array}{rrll}
 & S^1(a_1): & 0000\ 0000\ 0000\ 0000 & \\
\otimes & S^8(a_1): & 0000\ 0000\ 0000\ 0000 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_1 = 1) \\
 & S^2(a_1): & 0000\ 0000\ 0000\ 0000 & \\
\oplus & b_1: & 0000\ 0000\ 0000\ 0001 & \\
\hline
 & a_2: & 0000\ 0000\ 0000\ 0001 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_2): & 0000\ 0000\ 0000\ 0010 & \\
\otimes & S^8(a_2): & 0000\ 0001\ 0000\ 0000 & \\ \hline
 & & 0000\ 0000\ 0000\ 0000 & (p_2 = \frac{1}{4}) \\
 & S^2(a_2): & 0000\ 0000\ 0000\ 0100 & \\
\oplus & b_2: & 0000\ 0000\ 0000\ 0000 & \\ \hline
 & a_3: & 0000\ 0000\ 0000\ 0100 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_3): & 0000\ 0000\ 0000\ 1000 & \\
\otimes & S^8(a_3): & 0000\ 0100\ 0000\ 0000 & \\ \hline
 & & 0000\ 0000\ 0000\ 0000 & (p_3 = \frac{1}{4}) \\
 & S^2(a_3): & 0000\ 0000\ 0001\ 0000 & \\
\oplus & b_3: & 0000\ 0000\ 0000\ 0001 & \\ \hline
 & a_4: & 0000\ 0000\ 0001\ 0001 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_4): & 0000\ 0000\ 0010\ 0010 & \\
\otimes & S^8(a_4): & 0001\ 0001\ 0000\ 0000 & \\ \hline
 & & 0000\ 0000\ 0000\ 0000 & (p_4 = \frac{1}{16}) \\
 & S^2(a_4): & 0000\ 0000\ 0100\ 0100 & \\
\oplus & b_4: & 0000\ 0000\ 0000\ 0100 & \\ \hline
 & a_5: & 0000\ 0000\ 0100\ 0000 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_5): & 0000\ 0000\ 1000\ 0000 & \\
\otimes & S^8(a_5): & 0100\ 0000\ 0000\ 0000 & \\ \hline
 & & 0000\ 0000\ 0000\ 0000 & (p_5 = \frac{1}{4}) \\
 & S^2(a_5): & 0000\ 0001\ 0000\ 0100 & \\
\oplus & b_5: & 0000\ 0000\ 0001\ 0001 & \\ \hline
 & a_6: & 0000\ 0001\ 0001\ 0001 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_6): & 0000\ 0010\ 0010\ 0010 & \\
\otimes & S^8(a_6): & 0001\ 0001\ 0000\ 0001 & \\ \hline
 & & 0000\ 0000\ 0000\ 0000 & (p_6 = \frac{1}{64}) \\
 & S^2(a_6): & 0000\ 0100\ 0100\ 0100 & \\
\oplus & b_6: & 0000\ 0000\ 0100\ 0000 & \\ \hline
 & a_7: & 0000\ 0100\ 0000\ 0100 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_7): & 0000\ 1000\ 0000\ 1000 & \\
\otimes & S^8(a_7): & 0000\ 0100\ 0000\ 0100 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_7 = \frac{1}{16}) \\
 & S^2(a_7): & 0001\ 0000\ 0001\ 0000 & \\
\oplus & b_7: & 0000\ 0001\ 0001\ 0001 & \\
\hline
 & a_8: & 0001\ 0001\ 0000\ 0001 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_8): & 0010\ 0010\ 0000\ 0010 & \\
\otimes & S^8(a_8): & 0000\ 0001\ 0001\ 0001 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_8 = \frac{1}{64}) \\
 & S^2(a_8): & 0100\ 0100\ 0000\ 0100 & \\
\oplus & b_8: & 0000\ 0100\ 0000\ 0100 & \\
\hline
 & a_9: & 0100\ 0000\ 0000\ 0000 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_9): & 1000\ 0000\ 0000\ 0000 & \\
\otimes & S^8(a_9): & 0000\ 0000\ 0100\ 0000 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_9 = \frac{1}{4}) \\
 & S^2(a_9): & 0000\ 0000\ 0000\ 0001 & \\
\oplus & b_9: & 0001\ 0001\ 0000\ 0001 & \\
\hline
 & a_{10}: & 0001\ 0001\ 0000\ 0000 &
\end{array}$$

$$\begin{array}{crll}
 & S^1(a_{10}): & 0010\ 0010\ 0000\ 0000 & \\
\otimes & S^8(a_{10}): & 0000\ 0000\ 0001\ 0001 & \\
\hline
 & & 0000\ 0000\ 0000\ 0000 & (p_{10} = \frac{1}{16}) \\
 & S^2(a_{10}): & 0100\ 0100\ 0000\ 0000 & \\
\oplus & b_{10}: & 0100\ 0000\ 0000\ 0000 & \\
\hline
 & a_{11}: & 0000\ 0100\ 0000\ 0000 &
\end{array}$$

We thus have the differential trail

$T = (a_0\|b_0, a_1\|b_1, a_2\|b_2, a_3\|b_3, a_4\|b_4, a_5\|b_5, a_6\|b_6, a_7\|b_7, a_8\|b_8, a_9\|b_9, , a_{10}\|b_{10})$.

The differential probability of this trail is

$$\mathrm{DP}(T) \approx \Pi_{i=0}^{9}\ p_i = \frac{1}{4} \cdot 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{16} \cdot \frac{1}{4} \cdot \frac{1}{64} \cdot \frac{1}{16} \cdot \frac{1}{64} \cdot \frac{1}{4} = \frac{1}{2^{30}}.$$

Given $\mathcal{A} = 1$ in case that SIMON32/64$(a_0\|b_0)$ over 10 rounds results in

$a_{10}\|b_{10}$, and $\mathcal{A}=0$, if that is not the case, the advantage of $\mathcal{A}$ is:

$$\mathrm{Adv}_{\mathcal{A}} = \Big|\Pr[\mathcal{A}=1 \mid \text{SIMON32/64}_{10\text{ rounds}}] - \Pr[\mathcal{A}=1 \mid \text{RP}]\Big| = \frac{1}{2^{30}} - \frac{1}{2^{32}}.$$

In case of the differential distinguisher above, the online complexity M is 2, the offline complexity N is 0. To determine the security strength of this distinguisher $\mathcal{A}$, we use:

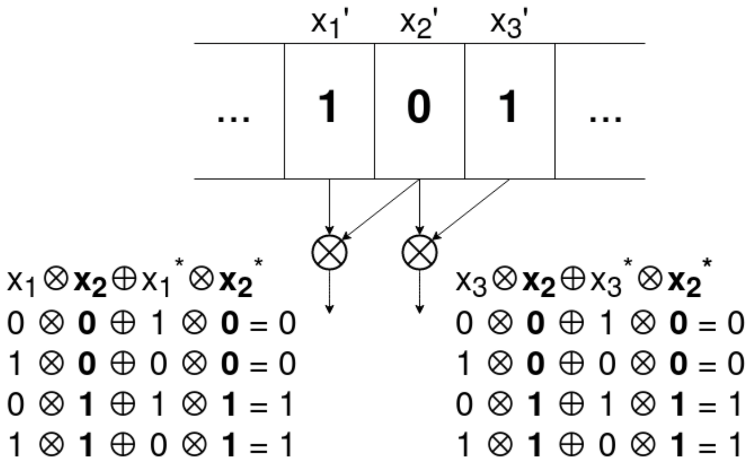
$$\log_2\left(\frac{N+M}{\mathrm{Adv}_{\mathcal{A}}}\right) = \log_2\left(\frac{0+2}{2^{-30}-2^{-32}}\right) \approx 31.415.$$

This gives an upper bound of 31.415 bits for the security strength of ten rounds of SIMON32/64. The security strength of exhaustive key search on SIMON32/64 is 64 bits (see Paragraph 3.4). The distuishing attack on ten rounds of SIMON32/64 in this paragraph is better than exhaustive key search on it, so SIMON32/64 reduced to ten rounds is considered broken.

5.2 Taking dependency of ANDs into account

In the previous paragraph, we assumed that in case of a bitwise multiplication, there is no dependency between the bits in the result. In reality, this usually is not true.

Figure 5.1:

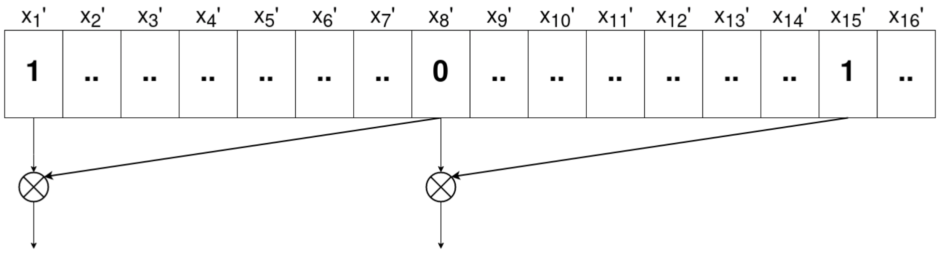


Take for example an input difference $x' = ...x'_1x'_2x'_3... = ...101...$ and the operation $x \otimes S^1(x)$. If the active ANDs (see Paragraph 4.2) given in Figure

5.1 are independent from each other, the probability that both result in 0, is $(\frac{1}{2})^2 = \frac{1}{4}$, as explained in Paragraph 4.2. However, we can see that x'_2 influences both ANDs. Since $x'_2 = 0$, the possibilities for the actual inputs are $x_2 = x^*_2 = 0$ and $x_2 = x^*_2 = 1$, where $x_2 \oplus x^*_2 = x'_2$. When $x_2 = x^*_2 = 0$, both ANDs will result in 0, as can be seen in Figure 5.1. When $x_2 = x^*_2 = 1$, both ANDs will result in 1. It is thus not possible that one of these ANDs will give the result 0 and the other the result 1. Both always will give the same result, which means they are dependent. Because of this, the probability that both ANDs will result in 0, is $\frac{1}{2}$, and not $(\frac{1}{2})^2$ in case of assumed independence.
As both of the 1-bits also each make another AND active, the total probability of having the result consist of only 0's is $(\frac{1}{2})^3$, and not $(\frac{1}{2})^4$ in case of assumed independence. In general it holds that $\mathrm{DP}((x101, 101y), 0000) = \frac{1}{8}$, where it is irrelevant what x and y are.

This cannot only be applied to 101-bit strings were the bits are directly next to each other, but also in the general case in which the bits each are k bits away from each other, and the operation $S^i(x') \otimes S^{i+k}(x')$ is applied. Take for example $k = 7$ and the multiplication $S^1(x') \otimes S^8(x')$ in the SIMON32/64 algorithm, where the size of x' is 16 bits, as depicted in Figure 5.2.

Figure 5.2:



If bit $x_1 = x_{15} = 1$ and bit $x_8 = 0$. The probability that both ANDs in Figure 5.2 will result in 0, is $\frac{1}{2}$, just as explained earlier for Figure 5.1.

Considering the dependency between the ANDs, we can calculate the DP for (ten rounds of) SIMON32/64 in a similar way as for the transformation χ [14]. Given $b = \chi(a)$, χ is defined by $b_i = a_i \oplus (a_{i+1} \oplus 1)a_{i+2}$, where $0 \leq i < l$, l is the block size of χ, and all indices all modulo l. For an input difference a' the bits of $b' = \chi(a) \oplus \chi(a \oplus a')$ are given by

$$b'_i = \chi(a')_i \oplus a'_{i+1}a_{i+2} \oplus a'_{i+2}a_{i+1}.$$

For this transformation, the weight w of the differential probability $\mathrm{DP}_\chi(a', b')$ is calculated by adding the Hamming weight h of a' and the number n of 001-patterns in a' [8]. So we have $\mathrm{DP}_\chi(a', b') = 2^{-(h+n)}$.

The differential probability of SIMON can be calculated in a similar way, which can be shown using the following lemmas: The proofs are based on the proofs in [14] which used [7].

Lemma 1 *Given $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $L\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$, where L is linear, it holds that for all $a', b' \in \mathbb{F}_2^n$, $\mathrm{DP}_f(a',b') = \mathrm{DP}_{f\circ L}(L^{-1}(a'),b') = \mathrm{DP}_{L\circ f}(a', L(b'))$.*

Proof.
$$\begin{aligned} F\mathrm{DP}_{f\circ L}(L^{-1}(a'),b') &= \mathrm{DP}_L(L^{-1}(a'),a') \cdot \mathrm{DP}_f(a',b') \\ &= \mathrm{DP}_f(a',b') \\ &= \mathrm{DP}_f(a',b') \cdot \mathrm{DP}_L(b', L(b')) \\ &= \mathrm{DP}_{f\circ L}(a', L(b')) \qquad \blacksquare \end{aligned}$$

Lemma 2 *Given $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $L\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$, $L(x) \mapsto Mx \oplus m$, where M is a $n \times n$ matrix and m is a n-bit vector, and $f \oplus L\colon \mathbb{F}_2^n \to \mathbb{F}_2^n, x \mapsto f(x) \oplus L(x)$, it holds that for all $a', b' \in \mathbb{F}_2^n$, $\mathrm{DP}_f(a',b') = \mathrm{DP}_{f\oplus L}(a', b' \oplus Ma')$.*

Proof. Let $f'(x) = f(x) \oplus L(x)$
$$\begin{aligned} \mathrm{DP}_f(a',b') &= \#\{x \in \mathbb{F}_2^n \mid f(x) \oplus f(x \oplus a') = b'\}/2^n \\ &= \#\{x \in \mathbb{F}_2^n \mid f'(x) \oplus L(x) \oplus f'(x \oplus a') \oplus L(x \oplus a') = b'\}/2^n \\ &= \#\{x \in \mathbb{F}_2^n \mid f'(x) \oplus Mx \oplus m \oplus f'(x \oplus a') \oplus Mx \oplus Ma' \oplus m = b'\}/2^n \\ &= \#\{x \in \mathbb{F}_2^n \mid f'(x) \oplus f'(x \oplus a') \oplus Ma' = b'\}/2^n \\ &= \#\{x \in \mathbb{F}_2^n \mid f'(x) \oplus f'(x \oplus a') = b' \oplus Ma'\}/2^n \\ &= \mathrm{DP}_{f'}(a', b' \oplus Ma') \\ &= \mathrm{DP}_{f\oplus L}(a', b' \oplus Ma') \qquad \blacksquare \end{aligned}$$

Given $\chi(x) = S^1(x) \otimes S^2(x) \oplus x \oplus S^2(x)$ and $C(x) = x \oplus S^2(x)$ and $\chi'(x) = \chi(x) \oplus C(x) = S^1(x) \otimes S^2(x)$ and using Lemma 2 (where $m = 0$), we find that
$$\mathrm{DP}_\chi(a',b') = \mathrm{DP}_{\chi'}(a', b' \oplus Ca').$$

Given $\chi'(x) = S^1(x) \otimes S^2(x)$ and given M where

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

such that

$$\begin{aligned}
&M(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15})^\intercal \\
=&(x_0, x_7, x_{14}, x_5, x_{12}, x_3, x_{10}, x_1, x_8, x_{15}, x_6, x_{13}, x_4, x_{11}, x_2, x_9)^\intercal
\end{aligned}$$

and SIMON32/64$'(x) = (S^7 \circ \chi' \circ M)(x) = S^1(x)S^8(x)$ and using Lemma 1, we find that

$$\mathrm{DP}_{\chi'}(a', b') = \mathrm{DP}_{\text{SIMON32/64}'}(M(a'), S^{-7}(b')).$$

Given SIMON32/64$'(x) = S^1(x) \otimes S^8(x)$ and $L(x) = S^2(x)$ and SIMON32/64$(x) = S^1(x) \otimes S^8(x) \oplus S^2(x)$ and using Lemma 2 (with m=0), we find that

$$\mathrm{DP}_{\text{SIMON32/64}'}(a', b') = \mathrm{DP}_{\text{SIMON32/64}}(a', b' + La').$$

$$\begin{aligned}
\text{So we get}\quad \mathrm{DP}_{\chi}(a', b') &= \mathrm{DP}_{\chi'}(a', b' \oplus Ca') \\
&= \mathrm{DP}_{\text{SIMON32/64}'}(M(a'), S^{-7}(b' \oplus Ca')) \\
&= \mathrm{DP}_{\text{SIMON32/64}}(M(a'), S^{-7}(b' \oplus Ca') \oplus L(M(a'))).
\end{aligned}$$

This means that we can calculate the DP of SIMON32/64 in a similar way as the DP of χ. Because there is no 101-pattern in the differential distinguisher in Paragraph 5.1, the result remains the same if we take dependent ANDs into account.

Chapter 6

Related Work

When Ray Beaulieu et. al. in 2013 published the specifications of the blockcipher SIMON [4], they initially did not give any security assessment. Hoda A. Alkhzaimi et al. [2] were the first to publish some cryptanalysis on SIMON using differential and impossible differential attacks for round reduced versions of the SIMON variants. An impossible differential attack uses differences that are impossible to obtain at a certain intermediate round. They showed a strong differential effect for the version 32/64 of the SIMON family and attacked 16 of the 32 rounds for this version.

Shortly afterwards, also in 2013, Farzaneh Abed et al [1] published some results of differential attacks on round reduced versions of SIMON. They attacked 18 rounds of SIMON32/64.

In 2015, Alex Biryukov et. al. [6] used a technique for automatic search for differential trails to improve previously reported trails and differentials for several versions of SIMON.

Tarun Yadav and Manoj Kumar [18] used a machine learning (ML) based differential distinguisher to apply differential cryptanalysis on SIMON in 2020.

Chapter 7

Conclusions

We applied a distinguishing attack on ten rounds of SIMON32/64 (Paragraph 5.1). The upper bound of the security strength of the used distinguisher is 31.415 bits. We found that this attack is better than exhaustive key search, which gives and upper bound of 64 bits for the security strength. For convenience, we first assumed that the ANDs are independent of each other. When we took into account that in fact these ANDs are dependent on each other, we did not find another result. Therefore, we conclude that SIMON32/64 reduced to ten rounds is broken.

7.1 Future work

Future work could be done on checking whether dependencies exist between round differentials. We could try to find multiple trails for the same differential. See for example Figure 7.1, where there are multiple possible routes between δ_0 and δ_3. If we only consider $(\delta_0, \delta_1, \delta_2, \delta_3)$, we find that $\mathrm{DP}(\delta_0, \delta_3)$ is at least $\frac{1}{4}$. If we also consider $(\delta_0, \delta_1', \delta_2, \delta_3)$, we find a higher DP for $(\delta_0.\delta_3)$, and could find a better attack. Another option would be to look for a better distinguisher by choosing a higher online complexity, so an M

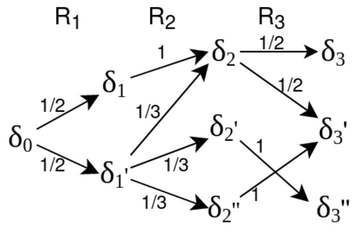


Figure 7.1: Example of multiple trails for a differential.

bigger than 2, by sending more queries. One could also seek for other distinguishers on more than ten rounds of SIMON32/64 or focus on other variants of SIMON.

Bibliography

[1] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential and Linear Cryptanalysis of Reduced-Round SIMON. 2013. https://eprint.iacr.org/2013/526.

[2] Hoda A. Alkhzaimi and Martin M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. 2013. https://eprint.iacr.org/2013/543.

[3] Amirki. Feistel cipher diagram. https://commons.wikimedia.org/wiki/File:Feistel_cipher_diagram_en.svg.

[4] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. 2013. https://eprint.iacr.org/2013/404.

[5] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. 2015. https://eprint.iacr.org/2015/585.

[6] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. 2015. https://link.springer.com/10.1007/978-3-662-46706-0_28.

[7] Anne Canteaut and Léo Perrin. On CCZ-equivalence, extended-affine equivalence, and function twisting. 2019. https://eprint.iacr.org/2018/713.

[8] Joan Daemen. Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis. 1995. https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf.

[9] Joan Daemen, Bart Mennink, and Jan Schoone. Lecture Notes Introduction to Cryptography. 2021.

[10] L. Batina J. Daemen. Slides of the course Cryptography. 2021.

[11] Biham E. and Shamir A. Differential cryptanalysis of the data encryption standard. 1993. New York: Springer Verlag. ISBN 978-0-387-97930-4.

[12] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. 1988. https://epubs.siam.org/doi/10.1137/0217022.

[13] Kattamuri Meghna. Block cipher modes of operation. https://www.geeksforgeeks.org/block-cipher-modes-of-operation/.

[14] Silvia Mella, Alireza Mehrdad, and Joan Daemen. Differential and Linear properties of vectorial boolean functions based on chi. 2023. https://www.researchgate.net/publication/370292931_Differential_and_Linear_properties_of_vectorial_boolean_functions_based_on_chi.

[15] NIST. FIPS 197, Advanced Encryption Standard (AES). 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[16] Ronald L. Rivest. The RC5 Encryption Algorithm. 1997. https://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf.

[17] Bruce Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). 1993. https://dl-acm-org.ru.idm.oclc.org/doi/10.5555/647930.740558.

[18] Tarun Yadav and Manoj Kumar. Differential-ML Distinguisher: Machine Learning based Generic Extension for Differential Cryptanalysis. 2020. https://eprint.iacr.org/2020/913.